



IBM Rational COBOL Runtime Guide for zSeries

Version 6 Release 0.1



IBM Rational COBOL Runtime Guide for zSeries

Version 6 Release 0.1

Note

Before using this information and the product it supports, read the information in "Notices" on page 239.

Sixth Edition (January 2012)

This edition applies to Version 6.0.1 of IBM Rational COBOL Runtime for zSeries (product number 5655-R29) and to all subsequent releases and modifications until otherwise indicated in new editions.

You can order publications through your IBM representative or the IBM branch office serving your locality.

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

© Copyright IBM Corporation 1994, 2012.

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

Contents

About This Document	vii
Who Should Use This Document	vii
Terminology Used in This Document	viii

Part 1. Preparing to Install. 1

Chapter 1. Preparing for the Installation of Rational COBOL Runtime	3
--------------------------------------------------------------------------------------	----------

Chapter 2. Storage Requirements for Rational COBOL Runtime	5
-----------------------------------------------------------------------------	----------

Virtual Storage Requirements	5
Rational COBOL Runtime Load Module Storage	5
Load Module Storage	5
COBOL Dynamic Storage	6
Rational COBOL Runtime Dynamic Storage	7
Storage Requirements for CICS	7
Disk Storage Requirements for Rational COBOL Runtime	8
Work Database Space For Segmented Programs.	8

Chapter 3. Installation Considerations.	9
--------------------------------------------------------	----------

z/OS Batch Considerations	9
DL/I Considerations.	9
DB2 Considerations	9
CICS Installation Considerations.	9
DL/I Considerations.	9
DB2 Considerations	10
Security Considerations	10
Monitoring and Tuning	10
CICS Utilities.	10
Client / Server Processing Considerations	10
Using the data Build Descriptor Option	11
Modifying CICS Resource Definitions.	11
IMS Installation Considerations.	12
IMS/ESA Exploitation	12
DB2 Considerations	12
Security Considerations	12
Monitoring and Tuning	13
IMS System Definition.	13
IMS Control Region	13
Work Database	13

Chapter 4. Customizing Rational COBOL Runtime	15
----------------------------------------------------------------	-----------

General Customization Considerations for z/OS	15
Customizing Rational COBOL Runtime	15
Security Considerations	15
Performance Considerations	15
Customizing Build Scripts	16
Modifying the Language Environment Runtime Option	16
Using Generated Programs with PL/I Programs	16

Installation and Language-Dependent Options for z/OS	16
Creating a custom conversion table	22
Changing the EGL System Libraries to Use Your Required Code Page	22

Part 2. Administering on z/OS Systems 25

Chapter 5. General System Considerations for z/OS Systems	27
----------------------------------------------------------------------------	-----------

Considerations that Affect Performance	27
Build Descriptor and Compiler Options	27
Modules in Memory	28
Files and Databases.	28
Defining and Loading VSAM Program Data Files.	28
Defining VSAM Data Sets	28
Loading Data in the Files.	30
Support for DBCS terminals	31
Extended Addressing Considerations for Rational COBOL Runtime	31
DB2 Considerations	32
Preparing Programs	32
Checking Access Authorization	32
Backing Up Data	32
Customizing Rational COBOL Runtime	32

Chapter 6. System Considerations for CICS.	33
-----------------------------------------------------------	-----------

Required File Descriptions	33
Segmented and Nonsegmented Processing	34
Using Transient Data Queues for Printing in z/OS CICS	35
z/OS CICS terminal printing	35
Special Parameter Group for the FZETPRT Program	36
CICS Entries for FZETPRT (DBCS only)	38
Using the New Copy Function	39
Specifying Recovery Options in CICS.	39
Considerations that Affect Performance	39
Residency (Modules in Memory) Considerations	39
Work Database Temporary Storage Queue Considerations	40
Terminal Printing	41
Using and Allocating Data Files in CICS.	41
Defining and Loading VSAM Data Files.	41
Using Remote Files	43
Defining Transient Data Queues	43
Considerations for Using DB2 in CICS	45
Associating DB2 Databases with CICS Transactions	45
Recovery and Database Integrity Considerations	45
Considerations for Using DL/I in CICS	45
Recovery and Database Integrity Considerations	45

Setting up the National Language 45

Chapter 7. System Considerations for z/OS Batch 47

Required File Descriptions 47
Using VSAM Program Data Files in z/OS Batch . . . 48
Considerations for Using DB2 in z/OS Batch . . . 48
 Recovery and Database Integrity Considerations 48
Considerations for Using DL/I in z/OS Batch . . . 48
 Defining the Program Specification Block (PSB) 48
 Recovery and Database Integrity Considerations 49
Considerations for Calling CICS programs from z/OS batch 49
Performance Considerations for z/OS Batch . . . 49
Runtime JCL 49

Chapter 8. System Considerations for IMS 51

Required File Descriptions 51
Defining the Program Specification Block (PSB) . . 52
Processing Modes 53
Printing Considerations for IMS 53
Recovery and Database Integrity Considerations . . 54
Considerations that Affect Performance 54
 Residency Considerations and the IMS Preload Function 54
 Database Performance 56
 Limiting MFS Control Blocks 56
 Monitoring and Tuning the IMS System 57
Considerations for Using DB2 in IMS. 57
 Recovery and Database Integrity Considerations 57
 Checking Authorization 57
Considerations for Using DL/I in IMS 58
 Recovery and Database Integrity Considerations 58
Maintaining the Work Database in IMS 58
 Deleting Old Records from the Work Database . 58
 Expanding the Work Database 60
 Supporting Multiple Work Databases. 63
Considerations for Message Format Services in IMS 64

Part 3. Preparing and Running Generated Applications 69

Chapter 9. Output of Program Generation on z/OS Systems 71

Allocating Preparation Data Sets 71
List of Program Preparation Steps after Program Generation 73
 Deploying generated code to USS 74
Output of Generation 74
 Objects Generated for Programs 77
 Link Edit File. 78
 CICS Entries 78
 Objects Generated for Data Tables 78
 Objects Generated for Form Groups 79

Chapter 10. z/OS Builds 81

z/OS Build Server 82
 Starting a z/OS Build Server 83

Starting a USS Build Server 85
Stopping servers. 85
Configuring a build server 85
Working with Build Scripts 85
 Working with z/OS Build Scripts 85
 Converting JCL to Pseudo-JCL 87

Chapter 11. Preparing and Running a Generated Program in CICS 91

Modifying CICS Resource Definitions 91
 Program Entries 91
 Transaction Entries 92
 Destination Control Table Entries 92
 File Control Table Entries. 93
 DB2 Entries 93
 Using Remote Programs, Transactions, or Files 93
CICS Setup for Calling CICS Programs from z/OS Batch 93
CICS Setup for Calling z/OS Batch Programs in CICS 93
Modifying CICS Startup JCL. 94
Making New Modules Available in the CICS Environment 94
Making Programs Resident 95
Running Programs under CICS. 95
 Starting the Transaction in CICS 95
 Controlling Diagnostic Information in the CICS Environment 95
 Printing Diagnostic Messages in the CICS Environment 95

Chapter 12. Creating or Modifying Runtime JCL on z/OS Systems 97

Tailoring JCL before Generation 97
Modifying Runtime JCL 98

Chapter 13. Preparing and Running Generated Programs in z/OS Batch . . . 101

Running Main Programs under z/OS Batch 101
Examples of Runtime JCL for z/OS Batch Programs. 101
 Running a Main Basic Program with No Database Access 102
 Running a Main Basic Program with DB2 Access. 102
 Running Main Basic Program with DL/I Access 103
 Running a Main Basic Program with DB2 and DL/I Access. 104
Recovery and Restart for z/OS Batch Programs 105

Chapter 14. Preparing and Running Generated Programs in IMS/VS and IMS BMP 107

Modifying the IMS System Definition Parameters 107
 Defining an Interactive Program 107
 Defining Parameters for a Main Basic Program as an MPP 108
 Defining Parameters for a Batch-Oriented BMP Program 109

Defining Parameters for a Transaction-Oriented BMP Program	109
Creating MFS Control Blocks	109
Making New Modules Available in the IMS Environment	110
Preloading Program, Print Services, and Data Table Modules	110
Running Programs under IMS	111
Starting a Main Program Directly	111
Starting a Main Transaction Program Using the /FORMAT Command	111
Running Transaction Programs as IMS MPPs	111
Running Main Basic Programs as MPPs	113
Running a Main Basic Program under IMS BMP	113
Examples of Runtime JCL for IMS BMP Programs	114
Running a Main Basic Program as an IMS BMP Program	114
Running a Main Basic Program as an IMS BMP Program with DB2 Access	115
Recovery and Restart for IMS BMP Programs	116

Chapter 15. Moving Prepared Programs to Other Systems from z/OS Systems 117

Moving Prepared Programs To Another z/OS System	117
Maintaining Backup Copies of Production Libraries	118

Part 4. Utilities 119

Chapter 16. Using Rational COBOL Runtime Utilities for z/OS CICS Systems 121

Using the CICS Utilities Menu	121
New Copy	122
Diagnostic Message Printing Utility	124
Diagnostic Control Options for z/OS CICS Systems	125
Using the Parameter Group Utility for z/OS CICS Systems	129

Chapter 17. Using Rational COBOL Runtime Utilities for IMS Systems 135

IMS Diagnostic Message Print Utility	135
------------------------------------------------	-----

Part 5. Diagnosing Problems 137

Chapter 18. Diagnosing Problems for Rational COBOL Runtime on z/OS Systems 139

Detecting Errors	139
Reporting Errors	139
Controlling Error Reporting	140
Error Reporting Summary	141
Using the Rational COBOL Runtime Error Panel	144
Printing Diagnostic Information for IMS	145
errorDestination Message Queue	145
IMS Log Format	146

Running the Diagnostic Print Utility	147
Printing Diagnostic Information for CICS	147
CICS Diagnostic Message Layout	147
Running the Diagnostic Print Utility	148
Analyzing Errors Detected while Running a Program	148

Chapter 19. Finding Information in Dumps 151

Rational COBOL Runtime ABEND Dumps	151
COBOL or Subsystem ABEND Dumps	151
Information in the Rational COBOL Runtime Control Block	152
Information in a Program, Print Services, or Data Table Profile Block	152
How to Find the Current Position in a Program at Time of Error	153

Chapter 20. Rational COBOL Runtime Trace Facility 155

Enabling EGL Program Source-Level Tracing with Build Descriptor Options	155
Activating a Trace	156
Activating a Trace Session for CICS or IMS/VS	156
Activating a Trace Session for z/OS Batch or IMS BMP	159
Deactivating a Trace Session	161
Printing Trace Output	161
Printing the Trace Output in CICS	161
Printing the Trace Output in IMS/VS	161
Printing the Trace Output in z/OS Batch or IMS BMP	161
Reporting Problems for Rational COBOL Runtime	161

Chapter 21. Common Messages during Preparation for z/OS Systems 163

Common Abend Codes during Preparation	163
MFS Generation Messages	163
DB2 Precompiler and Bind Messages	164
COBOL Compilation Messages	164

Chapter 22. Common System Error Codes for z/OS Systems 167

Common Error Codes	167
System Error Code Formats for sysVar.errorCode	167
Common System Error Codes in sysVar.errorCode	170
EGL Error Codes	171
Common SQL Codes	178
Common DL/I Status Codes	180
Common VSAM Status Codes	181
OPEN request type	181
CLOSE request type	181
GET/PUT/POINT/ERASE/CHECK/ENDREQ request types	182
COBOL Status Key Values	182

**Chapter 23. Rational COBOL Runtime
Return Codes, Abend Codes, and
Exception Codes 185**

Return Codes 185
ABEND Codes 185
 CICS Environments 185
 IMS, IMS BMP, and z/OS Batch Environments 187
 Exception Codes 188

**Chapter 24. Codes from Other
Products for z/OS Systems 191**

Common System Abend Codes for All
Environments 191
LE Runtime Messages 192
Common COBOL Abend Codes 193
Common IMS Runtime Messages. 193
Common IMS Runtime Abend Codes 194
Common CICS Runtime Messages 195

Common CICS Abend Codes 195
COBOL Abends under CICS 196

Part 6. Appendixes 197

**Appendix. Rational COBOL Runtime
Messages 199**

Message Format 199
ELA Messages 200
FZE messages 236
PRM messages 237

Notices 239

Trademarks 241

Index 243

About This Document

This manual provides information about customizing and administering Rational COBOL Runtime in the following environments:

- z/OS UNIX System Services (USS)
- z/OS® batch
- z/OS CICS®
- IMS/VS
- IMS™ BMP

It also provides information to enable you to prepare EGL programs for running in the z/OS environments.

For information about Java generation and runtimes for USS, refer to the *EGL Generation Guide*.

Note: Hereafter in this book, IBM® Rational COBOL Runtime for zSeries is referred to simply as “Rational COBOL Runtime.”

Who Should Use This Document

This manual is intended for system administrators and system programmers responsible for installing, maintaining, and administering Rational COBOL Runtime. It provides information to complete the following tasks:

- Manage system requirements
- Manage file utilization and conflicts

This manual is also intended for use by the programmers responsible for preparing and running EGL-generated programs. It provides information on the following items:

- Output of the generation process
- How to prepare generated programs for running
- Error codes
- How to use Rational COBOL Runtime utilities
- How to diagnose and report problems

Attention IBM VisualAge® Generator Users

Rational COBOL Runtime provides the required components to support development and execution of programs generated by Enterprise Generation Language (EGL) or VisualAge Generator Developer.

To understand how VisualAge Generator Developer is used with the Rational COBOL Runtime, refer to your VisualAge Generator documentation for information regarding the MVS™ environment. The VAGen MVS information also applies to the Rational COBOL Runtime when it is used in the z/OS environment.

Attention CICS Users

Refer to the CICS documentation for the level of CICS installed on your system for detailed information regarding CICS functions and operations.

Attention IMS Users

Refer to the IMS documentation for the level of IMS installed on your system for detailed information regarding IMS functions and operations.

Attention: Accessing EGL help

To access EGL help in the development workbench, click **Help→Help Contents** from the menu bar. When the help window appears, click **Developing→Developing EGL applications**.

Terminology Used in This Document

Unless otherwise noted in this publication, the following references apply:

- EGL refers to Enterprise Generation Language.
- CICS applies to Customer Information Control System.
- ELA.V6R0M1; represents the high-level qualifier used when Rational COBOL Runtime is installed.
- “CICS region” corresponds to CICS Transaction Server region.
- IMS/VS applies to Information Management System (IMS) and IMS Transaction Manager systems.
- IMS applies to IMS and IMS Transaction Manager, and to message processing program (MPP), IMS Fast Path (IFP), and batch message processing (BMP) regions. IMS/VS is used to distinguish MPP and IFP regions from the IMS BMP target environment.
- LE refers to Language Environment®.
- Workstation applies to a personal computer, not an AIX workstation.

Part 1. Preparing to Install

Chapter 1. Preparing for the Installation of Rational COBOL Runtime	3
----------------------------------------------------------------------------	---

Chapter 2. Storage Requirements for Rational COBOL Runtime.	5
Virtual Storage Requirements.	5
Rational COBOL Runtime Load Module Storage	5
Load Module Storage	5
COBOL Dynamic Storage	6
Rational COBOL Runtime Dynamic Storage	7
Storage Requirements for CICS	7
Disk Storage Requirements for Rational COBOL Runtime.	8
Work Database Space For Segmented Programs.	8

Chapter 3. Installation Considerations	9
z/OS Batch Considerations	9
DL/I Considerations.	9
DB2 Considerations	9
CICS Installation Considerations.	9
DL/I Considerations.	9
DB2 Considerations	10
Security Considerations	10
Monitoring and Tuning	10
CICS Utilities.	10
Client / Server Processing Considerations	10
Using the data Build Descriptor Option	11
Modifying CICS Resource Definitions.	11
APF authorization	11
Using Spool Files	11
Terminal Considerations	11
Temporary Storage	12
IMS Installation Considerations.	12
IMS/ESA Exploitation.	12
DB2 Considerations	12
Security Considerations	12
Monitoring and Tuning	13
IMS System Definition.	13
IMS Control Region	13
Work Database	13
DL/I Work Database Considerations	13
DB2 Work Database Considerations	13

Chapter 4. Customizing Rational COBOL Runtime	15
General Customization Considerations for z/OS	15
Customizing Rational COBOL Runtime	15
Security Considerations	15
Performance Considerations.	15
Customizing Build Scripts	16
Modifying the Language Environment Runtime Option	16
Using Generated Programs with PL/I Programs	16
Installation and Language-Dependent Options for z/OS	16

Creating a custom conversion table	22
Changing the EGL System Libraries to Use Your Required Code Page	22

Chapter 1. Preparing for the Installation of Rational COBOL Runtime

After selecting the production environments, do the following to prepare for the installation of the Rational COBOL Runtime:

- Obtain a copy of the *Program Directory for Rational COBOL Runtime for zSeries* (GI10-3377-00) (shipped with the product's installation materials).
- Determine the hardware, software, and storage requirements for the production environments selected.
- Install the hardware and software required by the Rational COBOL Runtime.
- Collect information before customization.
- Understand specific environment considerations before defining applications.

Before continuing with the current document, access the product website for details on product updates and prerequisites:

<http://www.ibm.com/developerworks/rational/products/rbde/>

Copies of documents are also available from the IBM Publications Center:

<http://www.elink.ibm.com/publications/servlet/pbi.wss>

There is also an EGL website and news group for EGL. The website is as at the following address:

<http://www.ibm.com/software/rational/cafe/community/egl/>

Chapter 2. Storage Requirements for Rational COBOL Runtime

The following sections give approximate estimates of Rational COBOL Runtime storage use by type of storage.

Virtual Storage Requirements

A program requires virtual storage for the following:

- Rational COBOL Runtime load modules
- Application load modules
- COBOL dynamic area
- Rational COBOL Runtime dynamic area

CICS programs also use specialized CICS storage facilities.

Rational COBOL Runtime Load Module Storage

Most of the modules in the runtime function are not linked with the generated programs. Only one copy of these modules needs to be available for use by all programs generated with Enterprise Generation Language (EGL).

For z/OS, these modules can be in a library (STEPLIB or DFHRPL), or placed in the link pack area (LPA). For CICS, you might want to make the modules resident. For IMS, you might want to preload the modules. Refer to the Rational COBOL Runtime program directory for a list of LPA eligible load modules.

Table 1. Rational COBOL Runtime Reentrant Load Module Storage Estimates

Function	Size	RMODE
CICS base services	240 KB	ANY
CICS base services, 24-bit addressing mode	8 KB	24
IMS/VS, IMS BMP, z/OS batch base services	255 KB	ANY
IMS/VS, IMS BMP, z/OS batch base services, 24-bit addressing mode	10 KB	24
Double-byte language ASCII/EBCDIC code conversion tables	Chinese - 50 KB	ANY

Load Module Storage

Load module storage is the storage required for generated COBOL programs. The load modules are created by link-editing the generated COBOL programs produced by EGL's COBOL generation facility. The size of the load module can be determined from the linkage editor module map. The size varies depending on the functions utilized with the programs.

The load module storage includes all generated programs, DataTable programs, FormGroup format modules, and print services programs used by a batch job step or transaction. The size of a load module also includes the small Rational COBOL Runtime programs that are statically linked with the programs. The load modules

produced by link-editing the generated programs are reentrant. Each module can be linked with RMODE(ANY) so that the load module can reside in extended storage.

The size of the Rational COBOL Runtime modules linked with each generated program, print services program, or DataTable program is shown in Table 2. These estimates should be added to the application load module size to determine the overall load module size.

Table 2. Rational COBOL Runtime Statically Linked Module Storage Estimates

Environment	Application	Print service program	DataTable program
CICS	2.5 KB	1 KB	1 KB
IMS/VS	1 KB	1 KB	1 KB
z/OS batch and IMS BMP	1.3 KB	1 KB	1 KB

Note: Rational COBOL Runtime modules are not statically linked with a FormGroup format module.

COBOL Dynamic Storage

Application load modules acquire dynamic storage while they are running. The COBOL runtime library requires this storage for application data structures such as records, forms, and DataTables. The storage includes both the internal and external data structures.

The COBOL **data** build descriptor option determines whether to acquire storage below the 16 MB line. The procedures shipped with the Rational COBOL Runtime enable **data** build descriptor option to control the value for the COBOL DATA compiler option. The default value of that build descriptor option is 31. Set **data** to 24 if an application calls another application or program that is linked as AMODE(24). DataTable program and print services programs must also use **data="24"** if any program being used is linked AMODE(24).

When you generate z/OS batch or CICS programs with dynamic storage requirements greater than 64 KB, the value data=31 is required.

The amount of storage required for internal data structures is listed in the compile listing of the COBOL application when the MAP, OFFSET, or LIST compiler options are used.

Applications that run outside of CICS use COBOL external data structures to share information between applications in the same run unit. The following table shows the storage estimates for external data structures.

Table 3. COBOL External Storage Utilization in Non-CICS Environments

Function	Storage Required
Rational COBOL Runtime control block	1KB
Environment is IMS/VS or IMS BMP	32 KB
IMS conversational processing	SPA size plus 18 bytes
File type SEQ, VSAM, GSAM, SMSGQ, MMSGQ or EZEPRINT SEQ, GSAM	96 bytes/file

Rational COBOL Runtime Dynamic Storage

When applications are running, Rational COBOL Runtime allocates storage as shown in Table 4. The initial program of the run unit determines where the shared storage between Rational COBOL Runtime and the generated COBOL program is allocated. If the initial program is generated with the **data** build descriptor option set to 24 or is link-edited with AMODE(24), this storage is allocated below the 16 MB line. Otherwise, the storage is allocated with 31-bit addresses as shown in the following table:

Table 4. Rational COBOL Runtime Dynamic Storage Utilization

Function	Storage Required	24- or 31-bit Addressing mode
Persistent dynamic storage pool. The pool is extended as needed in 32 KB increments. Most transactions or jobs require only the initial allocation. Segmented transactions in CICS or using a DB2® work database in IMS might require an extension.	32 KB increment	31
CICS - service program dynamic storage stack	48 KB	31
CICS with DL/I - DL/I buffers	64 KB	31
IMS/VS, IMS BMP, z/OS batch - service program dynamic storage stack	48 KB	24
IMS VS - DL/I buffers for path calls and DL/I work database	64 KB	based on data build descriptor option
IMS BMP - DL/I buffers for path calls and checkpoint input	96 KB	based on data build descriptor option
z/OS batch - DL/I buffers for path calls	64 KB	based on data build descriptor option
z/OS batch	64 KB	24

Storage Requirements for CICS

Generated COBOL applications use the following CICS storage facilities:

Table 5. Rational COBOL Runtime Use of CICS Storage Areas

Type of Storage	Function	Size
Transaction Work Area (TWA)	Rational COBOL Runtime Control Block. Offset in TWA is specified in twOffset build descriptor option.	1 KB
COMMAREA	Calls using COMMPTR	4 times the number of parameters
COMMAREA	Calls using COMMDATA	Total length of all parameters
COMMAREA	Remote calls	Total length of all parameters, plus 12
COMMAREA	transfer to program that passes a record	Length of record passed
COMMAREA	transfer to transaction or show statement that passes a record	Length of record passed plus 10

Table 5. Rational COBOL Runtime Use of CICS Storage Areas (continued)

Type of Storage	Function	Size
Shared storage	Shared DataTable contents, Shared DataTable control block	For each DataTable, length of DataTable contents plus: <ul style="list-style-type: none"> • 16 bytes for a message table • 8 bytes for other tables Also, one 50-byte record per shared DataTable.
Temporary storage queue (main or auxiliary)	Save information during converse or show statement	6 KB plus the length of all records and forms

Disk Storage Requirements for Rational COBOL Runtime

The auxiliary disk storage space required to install files for the Rational COBOL Runtime is approximately 2 MB. Additional disk space for user programs can vary.

Work Database Space For Segmented Programs

The space required for saving program status across a terminal I/O operation in CICS is the sum of all data areas (forms and records) for all segmented programs plus 6 KB per program. In CICS, disk space is used only if auxiliary temporary storage is specified as the work database during program generation.

The space required for saving program status across a terminal I/O operation in IMS/VS is the sum of the data areas (forms and records) for all segmented programs plus 4 KB per program.

For example, suppose that program A has the following:

- Two 4 KB records
- Two 512-byte forms
- 1 KB of working storage
- 100 terminals running application A in segmented mode

For CICS, the approximate required disk space is as follows:

$$(2 \times 4\,096 + 2 \times 512 + 1\,024 + 6\,144) \times 100 = 1\,638\,400$$

For IMS/VS, the approximate required disk space is

$$(2 \times 4\,096 + 2 \times 512 + 1\,024 + 4\,096) \times 100 = 1\,433\,600$$

If you are using a DL/I work database with IMS/VS, the storage required per terminal is inserted in 56 KB increments to localize access for all segments accessed on a single-path call. An additional 56 KB increment is required when help forms or extended error screens are used. A good estimate for work database size is 112 KB per active terminal.

Chapter 3. Installation Considerations

The following sections describe installation considerations for the Rational COBOL Runtime.

z/OS Batch Considerations

This section discusses some general considerations when installing EGL-generated programs in the z/OS batch environment.

DL/I Considerations

If the installation has programs that use DL/I databases, follow these steps:

1. Install the correct version of IMS. For more information on the correct version of IMS, see *Program Directory for Rational COBOL Runtime for zSeries*. This publication comes with the product or can be accessed from the IBM Publications Center at www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi.
2. Define databases and PSBs to IMS as described in the IMS utilities reference document.
3. Follow the optional DL/I-related steps for Rational COBOL Runtime installation as described in the *Program Directory for Rational COBOL Runtime for zSeries*.

DB2 Considerations

If the installation has programs that use relational databases, do the following:

1. Install the correct version of DB2. For more information on the correct version of DB2, see *Program Directory for Rational COBOL Runtime for zSeries*. This publication comes with the product or can be accessed from the IBM Publications Center at www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi.
2. Create the tables in the relational database that the programs will access.
3. Follow the optional DB2-related steps for Rational COBOL Runtime installation as described in the *Program Directory for Rational COBOL Runtime for zSeries*.
4. Define DB2 plans or packages as described in the DB2 installation and operation guides.

CICS Installation Considerations

This section discusses some general considerations when installing EGL-generated programs in the CICS environment.

DL/I Considerations

If the installation has programs that gain access to DL/I databases, you must do the following:

1. Install the correct version of IMS. For more information on the correct version of IMS, see *Program Directory for Rational COBOL Runtime for zSeries*. This publication comes with the product or can be accessed from the IBM Publications Center at www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi.

2. Define databases and PSBs to IMS as described in the IMS utilities reference document.
3. Follow the optional DL/I-related steps for Rational COBOL Runtime installation as described in the *Program Directory for Rational COBOL Runtime for zSeries*.
4. Add DL/I support to CICS and define databases and PSBs to CICS as described in the resource definition and installation and operation guides or in the IMS database control guide.

DB2 Considerations

If the installation has programs that gain access to relational databases, do the following:

1. Install the correct version of DB2. For more information on the correct version of DB2, see *Program Directory for Rational COBOL Runtime for zSeries*. This publication comes with the product or can be accessed from the IBM Publications Center at www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi.
2. Create the tables in the relational database that the programs use.
3. Follow the optional DB2-related steps for Rational COBOL Runtime installation as described in the *Program Directory for Rational COBOL Runtime for zSeries*.
4. Add DB2 support to CICS and define DB2 plans or packages to CICS as described in the DB2 system administration guides.

Security Considerations

CICS provides access control to resources (such as data files and programs) and transactions. This access can be controlled by the user or by the terminal.

CICS resources (such as data files, programs, destinations, journals, and temporary storage) can be assigned a security lock value. CICS users are assigned one or more key values. If a user is running a CICS transaction that is defined for resource security checking, the user's keys are checked every time a resource is requested. If the user does not have a key that matches the lock, access is denied by ending the transaction with an AEY7 ABEND code.

Monitoring and Tuning

Use CICS monitoring facilities to get information about CICS tasks.

Refer to the performance guide for your release of CICS for more information.

CICS Utilities

In the CICS environment, the Rational COBOL Runtime includes a set of utilities to assist in managing the error diagnosis and control facilities of the Rational COBOL Runtime environment. These utilities are EGL COBOL programs. See "Using the CICS Utilities Menu" on page 121 for more information about these utilities.

Client / Server Processing Considerations

EGL programs can use the benefits of client / server processing in the CICS environment. Client / server programs are developed like any other EGL program. Client / server processing is built on the `call`, `vgLib.startTransaction()`, and file I/O statements. You can define a program so that it calls a program on a remote CICS system. In addition, if the runtime environment is CICS, you can define a program so that it starts an asynchronous transaction on a remote CICS system or

gains access to a file on a remote CICS system. Refer to the **callLink**, **asynchLink**, and **fileLink** elements of linkage options part in the *EGL Generation Guide* for additional information about remote calls, remote asynchronous transactions, and remote file access.

Using the data Build Descriptor Option

Set the **data** build descriptor option to 24 on generated COBOL programs to enable calls from the generated program to programs using 24-bit addresses, as long as the length of the COBOL dynamic storage (as defined in the COBOL working-storage section) required for the application is less than 64 KB. Programs whose dynamic storage requirements are greater than 64 KB must be compiled with the **data** build descriptor option set to 31. Otherwise, COBOL ends the program with a 1009 ABEND code.

Note: The build scripts and procedures shipped with the Rational COBOL Runtime enables the **data** build descriptor option to control the value for the COBOL DATA compiler option. The **data** build descriptor option is set to 31 as the default for the CICS environment.

Modifying CICS Resource Definitions

CICS uses resource definitions to identify startup parameters, transactions, programs, files, databases, transient data destinations, and system locations for proper operation. The application developer must add or modify these definitions to correctly identify all objects to be used in the new or changed application.

To generate model resource definition online (RDO) program and transaction definitions, specify the **cicsEntries** build descriptor option with a value of RDO.

The CICS system initialization table needs to include EXEC=YES.

Add any transaction that invokes a program that uses DB2 to the resource control table (RCT) with the appropriate plan name. You can also use a resource definition.

APF authorization

For CICS environments, EGL Version 7 and above, you must add the distributed SELALMD load library to both the STEPLIB and the DFHRPL DD statement concatenation. This addition is needed because of the introduction of the new heap memory management modules. These new modules are loaded and run during an operating system call instead of an EXEC CICS call, which means they must be obtained from STEPLIB. Because you are adding these new memory management modules to STEPLIB, the SELALMD load library must become APF authorized; all STEPLIB load libraries must have this authorization. No special logic exists in SELALMD that requires APF authorization for its own sake.

Using Spool Files

To use the spool files, include the SPOOL=YES parameter in the System Initialization Table (SIT).

Terminal Considerations

Terminals used with EGL must have their alternate screen size either specified correctly in the alternate screen parameter of the TYPETERM definition, or omitted so the default of the primary screen size is used. An alternate screen size specification of (0,0) is not valid.

Any terminal defined as UCTRAN=YES in the TYPETERM definition and used for running pseudoconversational transactions might give different results than a terminal that is defined without UCTRAN=YES.

Any terminal used in a program that is the target of a **transfer to transaction** statement must have ATI=YES and TTI=YES specified in the TYPETERM definition.

Temporary Storage

Temporary storage queues used by the Rational COBOL Runtime must be defined as nonrecoverable. These queues start with X'EE'.

IMS Installation Considerations

This section discusses some general considerations when installing EGL-generated programs in the IMS environment.

IMS/ESA Exploitation

The build scripts shipped with the Rational COBOL Runtime cause the generated COBOL programs to be compiled with the **data="31"** build descriptor option and linked in AMODE(31) and RMODE(ANY). If the program calls another program that is linked with AMODE(24), then the **data="24"** build descriptor option is required.

You can link the generated COBOL program to run below the 24-bit line. However, if AMODE(24) is used to link the program, you must use the **data="24"** build descriptor option for the following situations:

- For a program that calls another program that is linked as AMODE(24)
- For the first program in the run unit, if any generated program in the run unit is linked as AMODE(24) or if a non-EGL program that uses DL/I is linked as AMODE(24)
- For a table or form services program, if any program being used is linked as AMODE(24)

DB2 Considerations

If the installation has programs that gain access to relational databases, do the following:

1. Install the correct version of DB2. For more information on the correct version of DB2, see *Program Directory for Rational COBOL Runtime for zSeries*. This publication comes with the product or can be accessed from the IBM Publications Center at www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi.
2. Create the tables in the relational database that the programs will access.
3. Follow the optional DB2-related steps for Rational COBOL Runtime installation as described in the *Program Directory for Rational COBOL Runtime for zSeries*.
4. Add DB2 support to IMS and define DB2 plans or packages to IMS as described in the DB2 system administration guide.

Security Considerations

Resource Access Control Facility (RACF[®]) can be used to control users authority to each transaction.

Monitoring and Tuning

Potential performance problems can be tracked before they occur by checking processing statistics on a regular basis. The following are some of the statistics to monitor:

- Use the IMS monitor facilities to check transaction utilization. Consider preloading applications or groups of applications that are frequently used.
- Use the IMS database monitor facilities to check how effectively the databases are performing and using space.

Refer to the IMS system administration document and the database administration guide for the release of IMS for additional information on monitoring the IMS online system and DL/I databases.

IMS System Definition

If you plan to use IMS, define all PSBs and transactions in the IMS system definition. In addition, define DL/I application databases.

IMS Control Region

You might need to review the values for the following:

- PSB work area pool (PSBW parameter)
- FORMAT pool (FBP parameter)
- MFS test area (MFS parameter)
- Communications input/output area (TPDP parameter)

In addition, if a DL/I work database is used, the work database must be added to either the control region JCL or to the dynamic allocation table.

Work Database

The work database is used to save the status of an EGL program during a **converse** statement, and to pass information during certain types of program-to-program message switches. The work database can be either a DL/I database or a DB2 table. The application developer specifies the **workDBType** build descriptor option when generating a program to determine which type of database is to be used. A DL/I or DB2 work database is used only for Rational COBOL transaction applications that are generated for the IMS/VS target environment. In general, a DL/I work database performs better than a DB/2 work database.

Multiple DL/I or DB2 work databases can be installed. Use separate databases for each application system to improve performance or data availability.

DL/I Work Database Considerations

If you plan to use a DL/I implementation for the work database, you might need to tailor the database description (DBD) before running the job that creates and initializes the DL/I work database.

DB2 Work Database Considerations

If you plan to use a DB2 implementation for the work database, review the database definition before running the job that initializes the DB2 work database. A DB2 synonym needs to be created for each user and program gaining access to the DB2 work database.

The DB2 work database requires a 32 KB page size. If a DB2 work database is used, you might need to increase the allocation of the 32 KB buffers. To increase

the allocation of buffers, modify and assemble the DB2 parameter module (default is DSNZPARM). Refer to the DB2 documents for the system for additional information.

If you select DB2, a DB2 plan for each transaction is needed even if the EGL program itself does not require DB2.

If you select DB2 and if the Rational COBOL Runtime needs maintenance applied to the module that handles the DB2 work database access, bind the DB2 plans again for all transactions that use this database.

There are also considerations with the DB2 authorization used by the IMS program that is gaining access to the DB2 work database. For example, authorization needs to be granted to LTERM and a synonym needs to be created.

Chapter 4. Customizing Rational COBOL Runtime

Before starting the customization process, determine the following:

- The target environments that application developers specify during generation
- Whether the programs use relational databases, hierarchical databases, or both.
- The IMS work database and terminal types
- The national language support requirements

General Customization Considerations for z/OS

The following sections discuss some general considerations for running EGL-generated programs in the supported z/OS environments.

Customizing Rational COBOL Runtime

Customizing Rational COBOL Runtime consists of performing some of the same procedures used to install the product on the system. These procedures are described in the *Program Directory for Rational COBOL Runtime for zSeries*.

Security Considerations

The Rational COBOL Runtime does not provide security services. Standard system or database manager security functions can be used with generated COBOL programs in the same way that they are used with customer-developed COBOL programs.

For example, if the EGL programs use DB2, define DB2 plans and give run authority to those users that are authorized to use the programs associated with the plan. The Resource Access Control Facility (RACF) can also be used to grant users authority to read or update files.

Performance Considerations

Other chapters in this book provide detailed information on considerations that affect performance. See the following chapters for information on these performance-related topics and others:

Performance Topic	Where to Find Info
Build descriptor options	<ul style="list-style-type: none">• Chapter 5, "General System Considerations for z/OS Systems," on page 27
Placing Rational COBOL Runtime product and generated application modules in memory	<ul style="list-style-type: none">• Chapter 5, "General System Considerations for z/OS Systems," on page 27
Residency and work-database considerations	<ul style="list-style-type: none">• Chapter 6, "System Considerations for CICS," on page 33• Chapter 8, "System Considerations for IMS," on page 51
Monitoring and tuning tools	<ul style="list-style-type: none">• Chapter 6, "System Considerations for CICS," on page 33• Chapter 8, "System Considerations for IMS," on page 51

Customizing Build Scripts

The Rational COBOL Runtime includes build scripts used for preparing generated programs for running. These build scripts can be customized to meet any data set naming conventions. Refer to the *EGL Generation Guide* for additional information.

Modifying the Language Environment Runtime Option

In the non-CICS environments, generated COBOL programs rely on COBOL working storage being initialized to binary zeros to determine whether COBOL Runtime is initialized. For Language Environment (LE), this is done by specifying `STORAGE=((00))` in the `CEEDOPT CSECT`.

The modified runtime options modules must be in a library allocated to the `STEPLIB` or placed in the link pack area or in a library managed by the Virtual Lookaside Facility and Library Lookaside features of z/OS for each non-CICS z/OS environment. If those modules are in a separate library, the library must precede the library that contains the unmodified modules.

Alternatively, these options can be set for each program by creating a `CEEUOPT` load module with these options set as listed above and link-editing this module with each generated COBOL program. Refer to the Language Environment documentation for more information on creating and using a `CEEUOPT` module to set runtime options.

Using Generated Programs with PL/I Programs

If PL/I programs are used with generated COBOL programs in a non-CICS environment, you must generate the COBOL program to invoke the PL/I program using a static COBOL call. This requires the PL/I programs to be linked with the COBOL program in the same load module.

If PL/I programs are used with generated COBOL programs in the CICS environment, you must generate the COBOL program to call the PL/I program using the `CICS LINK` command. This is the default linkage for the CICS environment. The calling and called programs must not be linked together for the CICS environment.

Refer to the *EGL Generation Guide* for additional information.

Installation and Language-Dependent Options for z/OS

The following are the installation options required for z/OS. To change the defaults, use the steps outlined in the *Program Directory for Rational COBOL Runtime for zSeries* (GI10-3377-00) to specify new settings. This document also provides instructions on customizing the Runtime Default Options and Language Dependent Options.

Table 6. Installation options for z/OS

Question	Default	Your Selection
Rational COBOL Runtime Default Options		
Default language code	ENU	_____
Bypass date edit on EOF	NO	_____
IMS/ESA® installed	NO	_____

Table 6. Installation options for z/OS (continued)

Question	Default	Your Selection
Rational COBOL Runtime trace buffer size	64	_____
CICS temporary storage control interval size	16	_____

The next table lists the national languages that are supported for these purposes:

- To present Rational COBOL Runtime messages on zSeries
- To present program-specific user messages based on the EGL `msgTablePrefix` property.

The code page for the language you specify must be loaded on your target platform.

Table 7. National language codes

Code	Languages
CHS	Simplified Chinese
CHT	Traditional Chinese
DES	Swiss German (for programs generated with VisualAge Generator)
DEU	German
ENP	Uppercase English (for programs generated with VisualAge Generator)
ENU	US English
ESP	Spanish
FRA	French
ITA	Italian
JPN	Japanese
KOR	Korean
PTB	Brazilian Portuguese

The following are the language-dependent options required for z/OS. One code is needed for each national language you install. The default values vary for each language.

Table 8. Rational COBOL Runtime National Language Dependent options for z/OS

Question	Default	Your Selection
National language code (US English)	ENU	_____
Long Gregorian date format	MM/DD/YYYY	_____
Short Gregorian date format	MM/DD/YY	_____
Long Julian date format	YYYY-DDD	_____
Short Julian date format	YY-DDD	_____
Conversion table name	ELACNENU	_____
Positive response character string	YES	_____

Table 8. Rational COBOL Runtime National Language Dependent options for z/OS (continued)

Question	Default	Your Selection
Negative response character string	NO	_____
Decimal point character*	.	_____
Numeric separator character*	,	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Simplified Chinese)	CHS	_____
Long Gregorian date format	YYYY-MM-DD	_____
Short Gregorian date format	YY-MM-DD	_____
Long Julian date format	YYYY-DDD	_____
Short Julian date format	YY-DDD	_____
Conversion table name	ELACNCHS	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	.	_____
Numeric separator character*	,	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Traditional Chinese)	CHT	_____
Long Gregorian date format	YYYY-MM-DD	_____
Short Gregorian date format	YY/MM/DD	_____
Long Julian date format	YYYY-DDD	_____
Short Julian date format	YY-DDD	_____
Conversion table name	ELACNCHT	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	.	_____
Numeric separator character*	,	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Swiss German)	DES	_____
Long Gregorian date format	DD.MM.YYYY	_____

Table 8. Rational COBOL Runtime National Language Dependent options for z/OS (continued)

Question	Default	Your Selection
Short Gregorian date format	DD.MM.YY	_____
Long Julian date format	YYYY.DDD	_____
Short Julian date format	YY.DDD	_____
Conversion table name	ELACNDES	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	,	_____
Numeric separator character*	.	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (German)	DEU	_____
Long Gregorian date format	DD.MM.YYYY	_____
Short Gregorian date format	DD.MM.YY	_____
Long Julian date format	DDD/YYYY	_____
Short Julian date format	DDD/YY	_____
Conversion table name	ELACNDEU	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	,	_____
Numeric separator character*	.	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (US English Upper Case)	ENP	_____
Long Gregorian date format	MM/DD/YYYY	_____
Short Gregorian date format	MM/DD/YY	_____
Long Julian date format	YYYY-DDD	_____
Short Julian date format	YY-DDD	_____
Conversion table name	ELACNENP	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	.	_____
Numeric separator character*	,	_____

Table 8. Rational COBOL Runtime National Language Dependent options for z/OS (continued)

Question	Default	Your Selection
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Spanish)	ESP	_____
Long Gregorian date format	DD/MM/YYYY	_____
Short Gregorian date format	DD/MM/YY	_____
Long Julian date format	DDD/YYY	_____
Short Julian date format	DDD/YY	_____
Conversion table name	ELACNESP	_____
Positive response character string	SI	_____
Negative response character string	NO	_____
Decimal point character*	,	_____
Numeric separator character*	.	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (French)	FRA	_____
Long Gregorian date format	MM/DD/YYYY	_____
Short Gregorian date format	MM/DD/YY	_____
Long Julian date format	DDD/YYYY	_____
Short Julian date format	DDD/YY	_____
Conversion table name	ELACNFRA	_____
Positive response character string	OUI	_____
Negative response character string	NAN	_____
Decimal point character*	,	_____
Numeric separator character*	.	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Italian)	ITA	_____
Long Gregorian date format	MM/DD/YYYY	_____
Short Gregorian date format	MM/DD/YY	_____
Long Julian date format	DDD/YYYY	_____
Short Julian date format	DDD/YY	_____

Table 8. Rational COBOL Runtime National Language Dependent options for z/OS (continued)

Question	Default	Your Selection
Conversion table name	ELACNITA	_____
Positive response character string	SI	_____
Negative response character string	NO	_____
Decimal point character*	,	_____
Numeric separator character*	.	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Japanese)	JPN	_____
Long Gregorian date format	YYY-MM-DD	_____
Short Gregorian date format	YY-MM-DD	_____
Long Julian date format	YYYY-DDD	_____
Short Julian date format	YY-DDD	_____
Conversion table name	ELACNJPN	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	.	_____
Numeric separator character*	,	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____
National language code (Korean)	KOR	_____
Long Gregorian date format	MM/DD/YYYY	_____
Short Gregorian date format	MM/DD/YY	_____
Long Julian date format	DDD/YYYY	_____
Short Julian date format	DDD/YY	_____
Conversion table name	ELACNKOR	_____
Positive response character string	YES	_____
Negative response character string	NO	_____
Decimal point character*	.	_____
Numeric separator character*	,	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____

Table 8. Rational COBOL Runtime National Language Dependent options for z/OS (continued)

Question	Default	Your Selection
National language code (Brazilian Portuguese)	PTB	_____
Long Gregorian date format	DD/MM/YYYY	_____
Short Gregorian date format	DD/MM/YY	_____
Long Julian date format	DDD/YYYY	_____
Short Julian date format	DDD/YY	_____
Conversion table name	ELACNPTB	_____
Positive response character string	SIM	_____
Negative response character string	NAO	_____
Decimal point character*	,	_____
Numeric separator character*	.	_____
Currency symbol	\$	_____
SQL host variable indicator	:	_____
SQL host column indicator	!	_____

* Decimal point and separator characters are determined by the **decimalSymbol** and **separatorSymbol** build descriptor options. In EGL programs that you generate for COBOL that do not use print forms, the default values for these options come from the language-dependent options module specified for your runtime installation. However, if you use print forms, the default value for the **decimalSymbol** option is a period, and the default value for the **separatorSymbol** option is a comma. If these values are not appropriate to your location, you must explicitly set these build descriptor options.

Upper case English (ENP) is also supported. It has the same defaults as ENU, except the conversion table name is ELACNENP.

Creating a custom conversion table

You might need a custom conversion table when your environment has minor differences from the environment for which a standard table was created.

1. Find the existing conversion table that is closest to your needs. For example, the ELACNCHS is used for simplified Chinese.
2. Make corrections to the file. The source is located in the AELASAMP library.
3. Assemble and link edit the module. A sample JCL to do this is in member ELACVPLK in the AELASAMP library.

Changing the EGL System Libraries to Use Your Required Code Page

The EGL runtime comes with 11 precompiled system library programs. These programs are written in COBOL and because they are distributed precompiled, they are using, by default, the English code page for any character to/from UNICODE transformations. This might not be acceptable for many users, and there is a way to alter this so that the runtime will use the code page that you require instead.

Each time a system library needs to perform a transformation between character and UNICODE, it calls a runtime program called EZEUCDE. This EZEUCDE program is written in COBOL and does any transformations using the COBOL intrinsic functions NATIONAL-OF and DISPLAY-OF. The source of EZEUCDE has been provided to you and is in your SELASAMP dataset. To alter this program so that it uses your required code page instead, all that needs to be done is for the source code to be recompiled with your code page specified in the COBOL parameters, and the resulting load module to either replace the EZEUCDE load module in the SELALMD dataset, or be placed in any dataset that will be ahead of SELALMD in the DD concatenation order. Sample JCL for this recompilation is available in your SELASAMP dataset, under the member name EZEUCDEJ.

Part 2. Administering on z/OS Systems

Chapter 5. General System Considerations for z/OS Systems 27

Considerations that Affect Performance	27
Build Descriptor and Compiler Options	27
Modules in Memory	28
Files and Databases.	28
Defining and Loading VSAM Program Data Files.	28
Defining VSAM Data Sets	28
Defining an Alternate Index	29
Loading Data in the Files.	30
Support for DBCS terminals	31
Extended Addressing Considerations for Rational COBOL Runtime	31
DB2 Considerations	32
Preparing Programs	32
Checking Access Authorization.	32
Backing Up Data	32
Customizing Rational COBOL Runtime	32

Chapter 6. System Considerations for CICS 33

Required File Descriptions	33
Segmented and Nonsegmented Processing	34
Using Transient Data Queues for Printing in z/OS CICS	35
z/OS CICS terminal printing	35
Special Parameter Group for the FZETPRT Program	36
PRTBUF Parameter.	37
PRTMPP Parameter.	37
PRTTYP Parameter	38
FORMFD Parameter	38
CICS Entries for FZETPRT (DBCS only)	38
Using the New Copy Function	39
Specifying Recovery Options in CICS.	39
Considerations that Affect Performance	39
Residency (Modules in Memory) Considerations	39
Virtual Storage Considerations and Residency	40
Work Database Temporary Storage Queue Considerations	40
Terminal Printing	41
Using and Allocating Data Files in CICS.	41
Defining and Loading VSAM Data Files.	41
Adding the Job Control Statements	41
Adding a CICS FILE Resource Definition for a File	42
Using Remote Files.	43
Defining Transient Data Queues	43
Defining Intrapartition Transient Data	44
Defining Extrapartition Transient Data	44
Considerations for Using DB2 in CICS	45
Associating DB2 Databases with CICS Transactions	45
Recovery and Database Integrity Considerations	45
Considerations for Using DL/I in CICS	45
Recovery and Database Integrity Considerations	45
Setting up the National Language	45

Chapter 7. System Considerations for z/OS Batch 47

Required File Descriptions	47
Using VSAM Program Data Files in z/OS Batch	48
Considerations for Using DB2 in z/OS Batch	48
Recovery and Database Integrity Considerations	48
Considerations for Using DL/I in z/OS Batch.	48
Defining the Program Specification Block (PSB)	48
Recovery and Database Integrity Considerations	49
Considerations for Calling CICS programs from z/OS batch	49
Performance Considerations for z/OS Batch	49
Runtime JCL	49

Chapter 8. System Considerations for IMS 51

Required File Descriptions	51
Defining the Program Specification Block (PSB)	52
Processing Modes	53
Printing Considerations for IMS	53
Recovery and Database Integrity Considerations	54
Considerations that Affect Performance	54
Residency Considerations and the IMS Preload Function	54
Preloading Rational COBOL Runtime Modules	55
Loading Rational COBOL Runtime Modules into the Link Pack Area	55
Preloading Generated Programs	56
Database Performance.	56
Limiting MFS Control Blocks	56
Monitoring and Tuning the IMS System.	57
Considerations for Using DB2 in IMS.	57
Recovery and Database Integrity Considerations	57
Checking Authorization	57
Considerations for Using DL/I in IMS	58
Recovery and Database Integrity Considerations	58
Maintaining the Work Database in IMS	58
Deleting Old Records from the Work Database	58
DL/I Work Database	59
DB2 Work Database	59
Expanding the Work Database	60
DL/I Work Database	60
DB2 Work Database	61
Supporting Multiple Work Databases.	63
DL/I Work Databases	63
DB2 Work Databases	63
Considerations for Message Format Services in IMS	64

Chapter 5. General System Considerations for z/OS Systems

This chapter describes the system requirements and considerations for administering the Rational COBOL Runtime in all of the supported z/OS environments.

This chapter contains the following topics:

- Considerations that affect performance
- Defining and loading VSAM program data files
- Support for DBCS terminals
- Extended addressing considerations for Rational COBOL Runtime
- DB2 considerations
- Backing up data
- Customizing Rational COBOL Runtime

Considerations that Affect Performance

Specifying certain build descriptor and compiler options and making reentrant programs resident in memory can affect the performance of EGL-generated programs.

Build Descriptor and Compiler Options

Setting the following build descriptor options may improve runtime performance:

- `checkIndices="NO"`
- `checkNumericOverflow="NO"`
- `fillWithNulls="NO"`
- `genReturnImmediate="YES"`
- `initIORecordsOnCall="NO"`
- `initNonIODataOnCall="NO"`
- `leftAlign="NO"`
- `math="COBOL"`
- `setFormItemFull="NO"`
- `spacesZero="NO"`
- `sqlErrorTrace="NO"`
- `sqlIOErrorTrace="NO"`
- `statementTrace="NO"`
- `validateMixedItems="NO"`
- `validateOnlyIfModified="YES"`
- `useXctlForTransfer="NO"`

Specifying the following compiler options also may improve runtime performance:

- NOSSRANGE.
- NOTEST.
- OPTIMIZE. OPTIMIZE provides faster runtime performance, but can significantly increase the compile time. Consider using the NOOPTIMIZE option during testing and the OPTIMIZE option when moving the program to production.

For details on COBOL compiler options, refer to your compiler documentation.

Setting the following build descriptor options may improve generation performance:

- `validateSQLStatements="NO"`
- `debugTrace="NO"`

Modules in Memory

Placing load modules in memory can improve performance by reducing the number of I/O operations (EXCPs). Load modules can be placed in memory by using the features of z/OS or the features of the environment in which you are running. Refer to the appropriate performance consideration sections for more detailed information about improving performance in a particular runtime environment.

General z/OS* methods to place load modules in memory are listed below:

- Place modules in the link pack area (LPA). Some of the modules that are shipped with the Rational COBOL Runtime are reentrant and can be placed in the LPA. Refer to the *Program Directory for Rational COBOL Runtime for zSeries* (GI10-3241-00) for information about modules that are reentrant and LPA eligible.

Generated programs, online print-service programs, FormGroup format modules, and shared DataTables are also reentrant and can be included in the LPA.

- Manage the Rational COBOL Runtime data sets and the data sets containing the generated programs, online print services programs, FormGroup format modules, and shared DataTables. Use the Virtual Lookaside Facility (VLF) and the Library Lookaside (LLA) features of z/OS. Those features can place both the load modules and the partitioned data set (PDS) directories in memory.

Note: The STEPLIB library is searched first. For the z/OS methods, the load module (for LPA) or the data set (for VLF/LLA) cannot be contained in the STEPLIB concatenation list.

Files and Databases

Standard tuning techniques (such as buffering) can be used with files and databases used by generated COBOL programs.

Defining and Loading VSAM Program Data Files

This section describes how to define and load VSAM data sets for use as program data files in the CICS, IMS BMP, or z/OS batch environment. The section contains the following information:

- Defining VSAM data sets
- Defining an alternate index
- Loading data into the files

Defining VSAM Data Sets

VSAM data files can be serial (ESDS), relative (RRDS), or indexed (KSDS) files. Use the IDCAMS program to define a user VSAM data file. Figure 1 on page 29 shows example JCL that can be used to define the VSAM files.

```

//DEFVSAM JOB ...
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN DD *

/* THE FOLLOWING SAMPLE DEFINES A */
/* VSAM INDEXED FILE */

DEFINE CLUSTER (NAME(ELA1.USER.KSDS) -
              VOL(xxxxxx) -
              CYLINDERS(pp ss) -
              KEYS(1 d) -
              RECORDSIZE(aaa mmm) -
              INDEXED)

/* THE FOLLOWING SAMPLE DEFINES A VSAM */
/* NUMBERED RELATIVE RECORD FILE */

DEFINE CLUSTER (NAME(ELA1.USER.RRDS) -
              VOL(xxxxxx) -
              CYLINDERS(pp ss) -
              RECORDSIZE(aaa mmm) -
              NUMBERED)

/* THE FOLLOWING SAMPLE DEFINES A VSAM */
/* ESDS FILE */

DEFINE CLUSTER (NAME(ELA1.USER.ESDS) -
              VOL(xxxxxx) -
              CYLINDERS(pp ss) -
              RECORDSIZE(aaa mmm) -
              NONINDEXED)

```

where:

xxxxxx Specifies a valid volume serial number

pp Specifies the primary number of cylinders to be allocated

ss Specifies the secondary number of cylinders to be allocated

l Specifies the length of the key

d Specifies the offset of the key

aaa Specifies the desired average record length

mmm Specifies the maximum record length

Figure 1. Defining VSAM Data Files

Defining an Alternate Index

An alternate index provides you with another way of gaining access to the records in a given KSDS file. Using a secondary key eliminates the need for you to keep several copies of the same information organized in different ways for different programs.

To gain access from an alternate index to the file through its prime index (base cluster), you must define a path to it. The path sets up an association between the alternate index and the base cluster, allowing the records in the data set to be available to you in different sequences. The alternate index is built after the base cluster is defined.

Figure 2 shows example IDCAMS definition commands for the base cluster and the alternate index cluster for an indexed file.

```
DEFINE CLUSTER (NAME(VSAM.KSDS.BASE.FILE) -  
  VOLUMES(xxxxxx) -  
  CYLINDERS(pp ss) -  
  KEYS(1 d) -  
  RECORDSIZE(aaa mmm) -  
  INDEXED)  
DEFINE ALTERNATEINDEX (NAME(VSAM.KSDS.ALT.INDEX) -  
  KEYS(1 d) -  
  CYLINDERS(pp ss) -  
  RELATE(VSAM.KSDS.BASE.FILE) -  
  VOLUMES(xxxxxx))  
DEFINE PATH(NAME(VSAM.KSDS.ALT.INDEX.PATH) -  
  PATHENTRY(VSAM.KSDS.ALT.INDEX))  
BLDINDEX INDATASET(VSAM.KSDS.BASE.FILE) -  
  OUTDATASET(VSAM.KSDS.ALT.INDEX)
```

where:

xxxxxx Specifies a valid volume serial number
pp Specifies the primary number of cylinders to be allocated
ss Specifies the secondary number of cylinders to be allocated
1 Specifies the key length
d Specifies the key displacement
aaa Specifies the desired average record length
mmm Specifies the maximum record length

Figure 2. Defining the Base Cluster and the Alternate Index Cluster

Loading Data in the Files

If you are using a VSAM indexed file (KSDS) and you want to open it for input only, initialize the file with at least one record. The file must have at least one record because a VSAM restriction prevents a file from being opened for input if the file is empty. While an empty file might be opened for output or both input and output, it must contain data to be opened for input.

There are several ways that you can put data into a file. One way is to create an EGL program that uses an **add** statement to add records to an empty serial file. Once the program ends, you can use the IDCAMS REPRO command to copy the serial file into an indexed file.

Another way is to write a program that uses an **add** statement to add records to an empty indexed file. You must close the file in order to make the new records accessible.

Another way to initialize a VSAM KSDS file is to use a utility program shipped with the Rational COBOL Runtime product. This utility can be used to initialize the key of a VSAM KSDS file. Figure 3 on page 31 shows how to initialize a VSAM KSDS file by setting the key to hexadecimal zeros.

```

//LOAD      JOB...
//JOBLIB    DD DSN=ELA.VxRxM0.SELALMD,DISP=SHR
//INITK     EXEC PGM=FZEZREBO,PARM='I,KSDS'
//SYSPRINT  DD SYSOUT=A,DCB=(LRECL=121,BLKSIZE=121,RECFM=FB)

//KSDS      DD DSN=USER.KSDS,DISP=SHR
//SYSIN     DD DUMMY

```

Figure 3. Initializing a VSAM KSDS File

You can also use the IDCAMS utility to load initial data into an indexed file. Figure 4 shows an example of loading data into a VSAM KSDS file. The data contained in the USER.KSDS.INPUT file is loaded into the USER.KSDS data set.

```

//JOB KSDSLOAD
//LOAD EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
      REPRO INDATASET('USER.KSDS.INPUT') OUTDATASET('USER.KSDS')
/*
//

```

Figure 4. Loading a VSAM KSDS File

Support for DBCS terminals

Rational COBOL Runtime provides support for the IBM Personal System/55 and the IBM 5550 family of terminals (emulating an IBM 3270 device). In addition to the basic hardware, this support uses character set F8 and four hardware attributes for double-byte character set (DBCS). The extended attributes are shift-out (SO) and shift-in (SI) enable, field outlining, color, and extended highlighting.

For the CICS environment, Rational COBOL Runtime sends hardware attributes to the terminal only if the terminal supports them. The attributes are ignored if the terminal does not support them.

The IMS environments use the Message Format Services (MFS) to support terminal and printer maps. During generation, you can use the **mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore** build descriptor options to specify device characteristics for all devices that are used in a FormGroup. Refer to the *EGL Generation Guide* for more details. Unpredictable results can occur if attributes are used that are not supported by the hardware. See “Considerations for Message Format Services in IMS” on page 64 for additional information concerning the message format services options.

Extended Addressing Considerations for Rational COBOL Runtime

Some of the code provided with Rational COBOL Runtime can run in extended addressing mode. This section describes considerations for using the extended addressing mode.

Most of the code shipped with Rational COBOL Runtime runs in 31-bit addressing mode and resides above the 16MB line.

Most of the storage acquired by Rational COBOL Runtime is above the 16MB line unless the first EGL program in the run unit is link-edited with AMODE(24) or generated with the **data** build descriptor option set to 24. The AMODE(24) program attribute specifies that the program runs in 24-bit addressing mode.

DB2 Considerations

This section discusses preparing programs and checking access authorization to database resources when using DB2 on z/OS systems.

Preparing Programs

Before running a program, the SQL statements need to be analyzed and prepared.

If you use DB2, you also need to bind the DB2 program plan.

Note: The above task is performed by the Rational COBOL Runtime build process.

If your programs run in the z/OS batch or IMS BMP environments, you might also need to tailor the runtime JCL templates. Refer to the *EGL Generation Guide* for additional information on tailoring runtime JCL templates.

Checking Access Authorization

The database manager checks whether program users have the authority to access tables or run programs. The type of checking done varies depending on your system and the processing mode.

When accessing DB2 in generated COBOL programs, program users must be authorized to run the corresponding DB2 program plan and package.

DB2 requires an authorization identifier to ensure that program users have the DB2 authority to perform operations on the database and tables. The type of authorization checking done depends on whether the processing mode is static or dynamic. The authorization identifier of the program developer performing the BIND command is used for static SQL statements; the authorization identifier of the program user is used for dynamic SQL statements. Generated COBOL programs use dynamic SQL statements in either of two cases:

- The SQL statement is in an EGL **prepare** statement
- The EGL statement uses an SQL record, and a host variable identifies the SQL table name associated with that record

Any other SQL statements in the program are static statements. Refer to the DB2 administration manual for more information on the various ways the authorization identifier value is set.

Backing Up Data

You should regularly back up your data. This includes all files related to Rational COBOL Runtime, private libraries, user-created data files, and user load libraries. System services are provided to back up and restore user libraries.

Customizing Rational COBOL Runtime

Customizing Rational COBOL Runtime consists of performing some of the same procedures used to install the product on the system. These procedures are described in the *Program Directory for Rational COBOL Runtime for zSeries* (GI10-3377-00). The program directory contains information on changing system options.

Chapter 6. System Considerations for CICS

This chapter provides additional system requirements and considerations for administering Rational COBOL Runtime in the CICS environment.

The following information is discussed:

- Required file descriptions
- Segmented and nonsegmented processing
- Using transient data queues for printing
- z/OS CICS terminal printing
- Using the new copy function
- Specifying recovery options in the CICS tables
- Considerations that affect performance
- Using and allocating data files
- Considerations for using DB2 in CICS
- Considerations for using DL/I in CICS
- Setting up the National Language

Required File Descriptions

Rational COBOL Runtime requires the following files:

File Name

Description

ELAD This transient data queue is the default destination for Rational COBOL Runtime error messages. Rational COBOL Runtime produces error messages when it detects an error that prevents a program from continuing.

The ELAD transient data queue is defined when Rational COBOL Runtime is installed. If you want to direct error messages for different transactions to different queues, define the other queues with the same characteristics as ELAD. Use the error diagnostic utility ELAC to direct error messages to the required queue. See the description of the utility in “Diagnostic Control Options for z/OS CICS Systems” on page 125 for more information.

ELACFIL

This is the error diagnostic control file. This file is created during customization.

ELAT This transient data queue is the destination for Rational COBOL Runtime trace records.

If requested, Rational COBOL Runtime can create trace records for selected runtime operations. The ELAT transient data queue is defined when Rational COBOL Runtime is installed. For details, see Chapter 20, “Rational COBOL Runtime Trace Facility,” on page 155.

ELATOUT

This file is associated with the ELAT transient data queue at installation time. The output of the Rational COBOL Runtime trace facility is sent to this data set. The attributes of this data set are DSORG=PS, LRECL=133, BLKSIZE=1330, RECFM=FBA.

EZEPRINT

The file that you associate to the Rational COBOL file name PRINTER at

resource association will be used when printing from a program that displays print forms. This file can be defined with a file type of SPOOL or TRANSIENT. This file is normally associated with the transient data queue PRIN.

If you installed Rational COBOL Runtime as described in the Rational COBOL Runtime program directory, PRIN is defined as an indirect destination associated with the system printer. The maximum record length that a generated program writes to the system printer is 650 bytes for double-byte character set (DBCS) print forms and 133 bytes for single-byte character set (SBCS) print forms. The first byte is an American National Standards printer control character. The DBCS record length is longer than the physical printer line length because the print record can contain outlining and shift-out/shift-in (SO/SI) control characters that do not appear on the device.

If you are using Rational COBOL Runtime to print to a file destination other than PRIN, the characteristics of that file should be the same as PRINTER.

EZEPRMG

This VSAM indexed file (KSDS) contains the parameter group records used for print control options for the Rational COBOL Runtime terminal printer utility, FZETPRT. The FZETPRT program reads this file searching for the parameter group matching the transaction name that started FZETPRT.

See “Special Parameter Group for the FZETPRT Program” on page 36 for a description of the print parameters. See “Using the Parameter Group Utility for z/OS CICS Systems” on page 129 for more information about maintaining this special parameter group.

Segmented and Nonsegmented Processing

Generated EGL textUI programs can issue a **converse** statement in either nonsegmented (CICS conversational) or segmented (CICS pseudoconversational) mode. When a **converse** statement is run in segmented mode or when a **show** statement is run, the current transaction ends and the program status is saved in a temporary storage queue until the terminal input is received. The **workDBType** build descriptor option specifies whether a main or auxiliary temporary storage queue is used. The temporary storage queues are deleted at the end of the run unit. The storage queue names have the following format:

xyyzt

where:

- x** Specifies a byte with the hex value X'EE'
- yyy** Specifies WRK (program working storage) or MSG (current form saved across help or error display)
- ttt** Specifies the terminal ID associated with the transaction

For details on segmentation, refer to the EGL help system.

Using Transient Data Queues for Printing in z/OS CICS

Printed output destined for a transient data queue is accumulated in temporary storage. The temporary storage queue name has the following format:

ttttnnnn

where:

tttt Is the transient data queue name
nnnn Is the EXEC Interface Block (EIB) task number

When a program ends, or a **close** statement is issued for a print map, or a segmentation break occurs, Rational COBOL Runtime enqueues on a transient data queue to prevent interspersed printing from other transactions. Rational COBOL Runtime copies the printed output onto the transient data queue. The printed output is in line character format with an American National Standards printer-control character.

The default print destination for z/OS CICS is a transient data queue named PRIN. If you installed Rational COBOL Runtime as described in the Rational COBOL Runtime program directory, PRIN is an indirect destination associated with the system printer. During program generation, this destination can be changed to any 4-character transient data queue name. The destination control table (DCT) entry for the queue determines the actual destination. The destination can be the system printer, a data set, or a terminal printer.

You can override the default destination at generation time by specifying the alternate destination as the system resource name for the **printer** file. You can change the print destination at run time by using the **converseVar.printerAssociation** system variable. Refer to the EGL help system for additional information on the **converseVar.printerAssociation** system variable.

EGL also provides a way of starting an asynchronous print task from a program and controlling the print destination from the program starting the asynchronous task. To do this, define the print task as a main basic program and generate it with the **printDestination="TERMINALID"** build descriptor option. Use the **vgLib.startTransaction()** system function to start the main basic program, specifying the print destination in the **vgLib.startTransaction()** parameters. The main basic program ignores the generated print destination and uses the destination specified in the **vgLib.startTransaction()** system function. Refer to the EGL help system for more information on the **vgLib.startTransaction()** system function.

z/OS CICS terminal printing

The program called FZETPRT supports terminal printing. This program runs as a CICS transaction that starts automatically when records are written to the transient data queue. If Rational COBOL Runtime was installed as described in the Rational COBOL Runtime program directory, the transaction name is EZEZ for IBM 5550-type printers and PRIN for all other printers. To send printed output to the terminal, you must include a **TYPE=INTRA** for the transient data queue in the CICS TDQUEUE resource definition. Specify PRIN or EZEZ for the transaction ID in the TDQUEUE resource definition entry. Unless you specify a terminal name in the TDQUEUE resource definition entry, the queue identifier must be the same as the terminal printer identifier. The trigger level in that entry must be set to 1 to ensure proper output. See "Printing Transient Data at a Terminal Device" on page 44 for a sample TDQUEUE resource definition entry.

When the FZETPRT program is initiated, it reads a line from the transient data queue, converts the American National Standards printer-control character to NL EOM format, and writes to the terminal printer specified in the DCT entry. The FZETPRT program buffers multiple print lines into a single CICS SEND command to improve performance.

When using terminal printing with Rational COBOL Runtime, you should be aware of potential problems regarding form-feed orders and page alignment. When the FZETPRT program is triggered, a form-feed order is issued to the printer to ensure that it begins printing at the top of a page. If a second form is sent to the queue before it is emptied by the FZETPRT program, a form-feed order is not issued before the second form is printed. Page alignment can vary depending on the timing with which successive forms are sent to the queue.

Another potential problem can occur when printing successive forms. If one of the forms in the series is defined with lines equal to, or one line fewer than, the lines-per-page setting on the printer, a blank page occurs between the printed forms. To avoid this, define the form size as 2 lines fewer than the lines-per-page setting on the printer. Because the FZETPRT program inserts a newline order to ensure that printing begins in column 1, the first line of the form to be printed is actually printed on the second line of the page. The second line must be allowed because a newline order is added after the last line of the form, which advances the print head to the beginning of the next line. If this happens to be the first line of the following page, the next form-feed order causes the page to be skipped before printing resumes.

Another thing to consider is that although Rational COBOL Runtime sometimes causes successive, stand-alone form-feed orders (“1”), the FZETPRT program suppresses all but one of these in converting them to NL EOM format.

If these form-feed considerations are too restrictive for your needs, consider using the FORMFD=NO parameter.

Special Parameter Group for the FZETPRT Program

You can provide terminal printing parameters to the FZETPRT program to vary the printed output by using a special parameter group file.

The FZETPRT program attempts to read a file named EZEPRMG for a parameter group that has the same name as the transaction used to start the FZETPRT program. For example, if the print transaction that starts the FZETPRT program is named PRIN, then FZETPRT tries to find the parameter group named PRIN. If the parameter group is not located in a file named EZEPRMG, or if EZEPRMG does not exist, then the FZETPRT program reads the DCAPRMG file to find the parameter group associated with this transaction.

When the transaction starts, the FZETPRT program reads the parameter group and varies the printer output according to the contents. If you need to use the terminal printing parameters, create a parameter group using the Rational COBOL Runtime utility provided for this purpose. See “Using the Parameter Group Utility for z/OS CICS Systems” on page 129 for more information about maintaining this special parameter group

For this parameter group, you can specify the following four parameters:

- PRTBUF=xxx
- PRTMPP=nnn

- PRTTYP=D
- FORMFD=NO

Note: Do not include blanks between keywords and their associated values.

PRTBUF Parameter

Use the PRTBUF parameter to set the size of the printer buffer. The number of SEND commands sent to the terminal printer depends on the size of the printer buffer. The following example shows how to specify the buffer size using the PRTBUF parameter:

```
PRTBUF=xxx
```

where:

xxx Is the size in bytes of the printer buffer

The FZETPRT program uses a default buffer size if any of the following conditions occur:

- The parameter is not specified in the parameter group.
- There is no parameter group associated with the transaction.
- The parameter keyword is misspelled.
- The value specified is not valid (values greater than 8K bytes, smaller than 480 bytes, or not numeric).
- The EZEPRMG or DCAPRMG file does not exist or is not available.

The default buffer size is 2KB (where KB equals 1024 bytes) for the standard character set printers and 480 bytes for LU type 3 printers.

For double-byte character set (DBCS) users the default buffer size and the maximum buffer size allowed is 1918 bytes. The default value is used if your specified value exceeds the maximum number of bytes.

When the buffer size is larger than the default, usage of the PRTBUF parameter is optional. However, using the PRTBUF parameter is recommended to reduce the number of SEND commands sent to the terminal. If the printer buffer size is smaller than the default, specify the real buffer size using this parameter. Not specifying the real buffer size can cause unpredictable results.

PRTMPP Parameter

Use the PRTMPP parameter to set the maximum number of print positions. The following example shows how to specify the number of print positions using the PRTMPP parameter:

```
PRTMPP=nnn
```

where:

nnn Is the physical length (maximum print position) of the printer line

The FZETPRT program assumes a default maximum print positions of 132 if any of the following occurs:

- The parameter is not specified in the parameter group.
- There is no parameter group associated with the transaction.
- The parameter keyword is misspelled.
- The value specified is not valid (not numeric).
- The EZEPRMG or DCAPRMG file does not exist or is not available.

Use caution when coding the value of this parameter. If the value entered is a valid numeric, the FZETPRT program uses the value without validating it. If the value is greater than the number of print positions available on the actual printer, possible malfunctioning can take place causing more line skips than necessary.

Note: For DBCS users, this parameter must be specified unless the printer is configured with MPP=132.

PRTTYP Parameter

Use the PRTTYP parameter if you use a DBCS printer. The following example shows how to specify the use of a DBCS printer using the PRTTYP parameter:

```
PRTTYP=D
```

Note: This parameter must be used to specify that you are a DBCS user and your output is being directed to an IBM 5550-family printer.

If you use multiple printers with different characteristics (namely different MPP, different buffer size, or DBCS versus non-DBCS printers), you need as many transaction IDs as there are printers, each one associated with the FZETPRT program. For examples of table entries for two printers, see the CICS transaction definitions provided with Rational COBOL Runtime for the PRIN (non-DBCS printers) and EZEZ (DBCS printers) transactions.

FORMFD Parameter

Use the FORMFD parameter to control the form-feed orders that the FZETPRT program issues. The following example shows the format of the FORMFD parameter:

```
FORMFD=NO
```

The FZETPRT program defaults to inserting form-feed orders into the printer data stream if any of the following occurs:

- The parameter is not specified in the parameter group.
- There is no parameter group associated with the transaction.
- The parameter does not appear as FORMFD=NO.
- The EZEPRMG or DCAPRMG file does not exist or is not available.

If the parameter is specified correctly, the FZETPRT program does not insert form-feed orders for any reason. This includes using the **converseLib.pageEject** system function, closing the printer, or the initial form feed that is normally done. All forms control depends on the map size specified during map definition.

CICS Entries for FZETPRT (DBCS only)

If you are using an SCS-type printer and you use DBCS, ensure that your system programmer has coded the destination control table (DCT) and the program control table (PCT) entries for a transaction that runs FZETPRT with the following option:

```
MSGPOPT=CCONTRL
```

The MSGPOPT option defines the optional facilities that a task can use. The CCONTRL parameter indicates that the program can control the outbound chaining of request units. Refer to the CICS manuals for more information.

Using the New Copy Function

The new copy function (either the Rational COBOL Runtime new copy utility or the CICS NEWCOPY command) causes a transaction to use a new copy of a program, FormGroup, or DataTable referenced in the transaction. For the purposes of this function, libraries and services are considered to be programs. The Rational COBOL Runtime new copy utility is implemented as an EGL program in the CICS environment. Active transactions continue to use the current version of a program, FormGroup, or DataTable until the transaction either completes or reaches the end of a segment. A new copy of the program, FormGroup, or DataTable is then made available to the transaction by Rational COBOL Runtime. Use the new copy function when programs, FormGroups, and DataTables are modified and generated again. This enables you to install new versions of programs, FormGroups, and DataTables onto your system without disrupting operation.

For programs and FormGroups you can use the CICS NEWCOPY command or the Rational COBOL Runtime new copy utility to cause the new copy of the program to be used the next time a load request is issued for the program.

The Rational COBOL Runtime new copy utility does a new copy for both the online print services program and the FormGroup format module when you specify a part type of FormGroup. If you use the CICS NEWCOPY command for a FormGroup, you must issue the NEWCOPY for both the online print services program and the FormGroup format module.

For DataTables, you must use the Rational COBOL Runtime new copy utility to cause a fresh copy of the DataTable to be used the next time a load request is issued for the DataTable. Do not use the CICS NEWCOPY command for DataTables. The Rational COBOL Runtime new copy utility sets a flag indicating that the new copy of the table is to be used the next time a program loads the table contents.

For more information on the Rational COBOL Runtime new copy utility, see “New Copy” on page 122.

Specifying Recovery Options in CICS

EGL-generated programs can make use of all the z/OS CICS recovery and data integrity features. For a description of those features, refer to the recovery and restart information for your release of CICS.

Considerations that Affect Performance

This section describes factors that affect system performance and suggestions on how to improve performance. For information beyond what is stated in this section, refer to the performance guide for your release of CICS.

Residency (Modules in Memory) Considerations

The performance of a program is affected by the number of times that a running program requires access to a disk. Programs require access to disks for the following reasons:

- Locating and loading Rational COBOL Runtime load modules
- Retrieving and storing user data
- Locating and loading application programs, FormGroup format modules and online print services programs, and DataTable programs

The Rational COBOL Runtime loads objects as they are needed. For example, the Rational COBOL Runtime loads a program, library, service, online print services program, FormGroup format module, or DataTable when another program calls or references it. If you make an object resident, then the object remains in storage after it is loaded by the Rational COBOL Runtime. You can use the RES parameter on the program definition to make any of these resident: a program, library, service, online print services program, or FormGroup format module.

For DataTables, use the **shared** and **resident** properties in the DataTable part definition to control residency for all programs that use the DataTable. In addition, in VisualAge Generator Compatibility mode, you can use the **deleteAfterUse** property on the program's **use** declaration for the DataTable to affect how the program manages the DataTable.

Virtual Storage Considerations and Residency

It is true that if a program, library, service, online print services program, FormGroup module, or DataTable program is resident, less I/O is required for multiple loads. However, making these objects resident requires more virtual storage because the modules accumulate in storage as they are loaded and are not deleted after they are used.

When deciding what to make resident, consider the following:

- Storage constraints
- Frequency of program use
- Long running programs versus programs that are started more frequently

Because most systems have virtual storage constraints, it is not possible to make everything resident. You should establish priorities for deciding which objects you want to make resident. These residency priorities reflect a trade-off between program usage and storage constraints. Your priorities can dictate that some components of a program (such as the online print services program or FormGroup format module) should be made resident, while other components (such as DataTables) should not.

In CICS, when a program component is made resident, it remains in storage from the time it is loaded into storage until either CICS is shut down or the new copy function is used. To aid in deciding which programs should be made resident, you can use CICS shutdown statistics to determine how often a generated program or other component is loaded into the region or partition.

Generally, objects that are loaded more than once are prime candidates for residency. Examples of this a DataTable that is used by more than one program or a program that is called more than once.

Programs that are not frequently initiated or have long running time should not be made resident.

If you plan to run a program in segmented mode (CICS pseudoconversational), you should consider making all components of the program resident. In pseudoconversational mode, the program and its components are deleted and are loaded again at each segment break if they are not made resident, and these actions degrade performance.

Work Database Temporary Storage Queue Considerations

When running in pseudoconversational mode (using a segmented **converse** statement), the data and the status associated with the program must be saved

during user think time. You use the **workDBType** build descriptor option to control whether this information is saved into the CICS main temporary storage or auxiliary storage. Using main temporary storage can result in better performance because the data is written to memory within the CICS address space instead of writing the data to disk space.

Note: Use of main temporary storage can degrade system performance because the increased address space that is referenced can increase the paging activity. Also, CICS can experience a short-on-storage condition if the program data to be saved exceeds the available CICS storage. Therefore, if you take advantage of main temporary storage for programs requiring better performance, you should monitor your system to ensure that virtual storage problems do not occur.

The amount of data written or read on each request to CICS when saving program data and status, can also affect performance. The installation options module, ELARPIOP, specifies the largest size record Rational COBOL Runtime writes to main or auxiliary temporary storage. The default size is 32KB (where KB equals 1024 bytes), which is the largest value allowed by CICS. Use a large value to ensure that the least number of write requests are required, and, if using auxiliary storage, to ensure that the least number of I/O operations are required. See the *Program Directory for Rational COBOL Runtime for zSeries* for information on how to change the value in the installation options module.

Note: If you are using auxiliary storage queues, you should ensure the control interval size (CISIZE) of the VSAM data set used for auxiliary temporary storage matches the size specified in the installation options file. If the CISIZE for the data set is smaller, CICS splits the data written or read into smaller pieces and does multiple I/O operations for each Rational COBOL Runtime request. Also ensure that you have an adequate number of buffers for the auxiliary temporary storage data set in order to reduce the number of physical I/O operations.

Terminal Printing

The performance of terminal printing can be enhanced by specifying the PRTBUF parameter for the FZETPRT program. See “z/OS CICS terminal printing” on page 35 for more information on terminal printing and the PRTBUF parameter

Using and Allocating Data Files in CICS

This section describes how to define data files for use in generated EGL-generated programs in the CICS environment.

Defining and Loading VSAM Data Files

Before CICS programs can use VSAM data files, you must define and load them. See “Defining and Loading VSAM Program Data Files” on page 28 for information on defining VSAM data sets, defining an alternate index, and loading a VSAM data set.

Adding the Job Control Statements

After the data set has been defined and loaded, add the data set name to the CICS startup JCL to allocate user files. You can also let CICS dynamically allocate the data set to the file using the information specified in the CICS FILE resource definition. Figure 5 on page 42 shows example allocation statements for an

indexed, relative, and serial file, and an alternate index.

```
//KSDSFILE DD DSN=ELA1.USER.KSDS,DISP=SHR
//RRDSFILE DD DSN=ELA1.USER.RRDS,DISP=SHR
//ESDSFILE DD DSN=ELA1.USER.ESDS,DISP=SHR
//KSDSAIX DD DSN=VSAM.KSDS.ALT.INDEX.PATH,DISP=SHR
```

Figure 5. Allocating User Files

Adding a CICS FILE Resource Definition for a File

After you have defined and loaded the data set and added it to the CICS startup JCL, you must also create a CICS FILE resource definition entry so that the CICS program can access the data set. Use the CICS Resource Definition Online (RDO) to create the FILE resource definition.

Figure 6 on page 43 shows resource definitions that can be used to add a file name. Rational COBOL Runtime uses the name on the FILE operand. The FILE operand name must be the same as the DD name in the CICS startup JCL. All other operands must be the same as when you create a FILE resource definition for an indexed, relative, or serial file that is accessed by a non-EGL program.

Create a FILE resource definition for every file used by a program. You can define the files as remote files.

For further information, refer to the appropriate CICS resource definition guide for your environment.

```

KSDS

DEFINE FILE(KSDSFILE) GROUP(xxxxxx)
    DSNAME(Indexed.DSName)
    DISPOSITION(SHARE) ADD(YES)
    BROWSE(YES) DELETE(YES) READ(YES)
    UPDATE(NO) RECORDFORMAT(F)
    STRINGS(8) LSRPOOLID(NONE)
    RECOVERY(NONE) NSRGROUP(GROUP1)
    INDEXBUFFERS(8) DATABUFFERS(9)

Alternate Index

DEFINE FILE(KSDSAIX) GROUP(xxxxxx)
    DSNAME(AlternateIndex.DSName)
    LSRPOOLID(NONE) DISPOSITION(SHARE)
    STRINGS(5) NSRGROUP(GROUP1)
    BROWSE(YES) DELETE(NO) READ(YES)
    ADD(NO) UPDATE(NO) RECORDFORMAT(F)
    RECOVERY(NONE) INDEXBUFFERS(5)
    DATABUFFERS(6)

RSDS

DEFINE FILE(RSDSFILE) GROUP(xxxxxx)
    DSNAME(Relative.DSName)
    DISPOSITION(SHARE) ADD(YES)
    BROWSE(YES) DELETE(YES) READ(YES)
    UPDATE(NO) RECORDFORMAT(F)
    STRINGS(8) LSRPOOLID(NONE)
    RECOVERY(NONE) NSRGROUP(GROUP1)
    INDEXBUFFERS(8) DATABUFFERS(9)

ESDS

DEFINE FILE(ESDSFILE) GROUP(xxxxxx)
    DSNAME(EntrySequenced.DSName)
    DISPOSITION(SHARE) ADD(YES)
    BROWSE(YES) DELETE(YES) READ(YES)
    UPDATE(NO) RECORDFORMAT(F)
    STRINGS(8) LSRPOOLID(NONE)
    RECOVERY(NONE) NSRGROUP(GROUP1)
    INDEXBUFFERS(8) DATABUFFERS(9)

```

Figure 6. Adding a File Resource Definition

Using Remote Files

EGL-generated programs can access files that do not reside on your CICS system.

Refer to the EGL online help for additional information on the **fileLink** element of the linkage options part. Refer to the appropriate CICS manuals for information about defining remote programs, transactions, or files.

Defining Transient Data Queues

Transient data queues are used in CICS for reading or writing data from tapes, disks, or other sequential files. If you associated a serial file with a transient data queue at generation, you must define a CICS TDQUEUE resource definition for the queue.

You can define the following types of transient data queues:

- Intrapartition (temporary data)
- Extrapartition (data that other non-CICS regions can use)

Intrapartition transient data files contain data that is not usable after it is read.

Defining Intrapartition Transient Data

The following two examples show how to define intrapartition transient data files.

Passing Transient Data between CICS Transactions: This is an example of a TDQUEUE resource definition that can be used to pass data from one CICS transaction to another. The file destination specified at generation in the resource association part should be **systemName="xxxx"**.

```
DEFINE TDQUEUE(xxxx) GROUP(gggggggg)
      TYPE(INTRA) ATIFACILITY(FILE)
```

Printing Transient Data at a Terminal Device: This is an example of a TDQUEUE resource definition that can be used for terminal printing in Rational COBOL Runtime. At generation time, the resource associations part specifies how you want to handle *printer*. The default is the first four characters, for example, *prin*. (A TDQUEUE resource definition is supplied for *prin* that sends the printed output to the system printer.) The program supplied for printing, FZETPRT, reads records from the transient data queue and issues SEND commands to the terminal in order to print the records.

In this sample TDQUEUE entry, the PR01 terminal is to receive the printed output. PR01 is a z/OS CICS printer terminal name. You specify the *printer* destination at generation as PR01. Rational COBOL Runtime writes the printed output to the transient data queue, PR01. The transaction PRIN starts and causes the program FZETPRT to run. The data is read from the transient data queue and sent to the terminal, PR01. The RDO TRANSACTION entry for PRIN and the PROGRAM entry for FZETPRT are supplied. You must supply the destination control table and the terminal control table entries for the transient data and terminal.

```
DEFINE TDQUEUE(PR01) GROUP(gggggggg)
      TYPE(INTRA) ATIFACILITY(TERMINAL)
      TRANSID(PRIN) TRIGGERLEVEL(1)
```

If the terminal printer is a DBCS printer, specify EZEZ as the TRANSID.

Defining Extrapartition Transient Data

Data to be read from tape or sent to a printer is contained in extrapartition transient data queues.

The following example shows how to use extrapartition transient data queues. These files can be used by non-CICS devices and by CICS.

Printing Transient Data: This is an example of a TDQUEUE resource definition specification that can be used to print output on a high-speed system printer. The file destination specified at generation in the resource associations part should be **systemName="xxxx"**.

```
DEFINE TDQUEUE(zzzz) GROUP(gggggggg)
      TYPE(EXTRA) TYPEFILE(OUTPUT)
      RECORDFORMAT(VARIABLE) BLOCKFORMAT(BLOCKED)
      RECORDSIZE(133) BLOCKSIZE(1330)
```

You also need to add the appropriate DD statement to the CICS runtime JCL to assign a printer to the file name. The extrapartition destination data queue sample shown above requires the following DD statement:

Considerations for Using DB2 in CICS

This section presents considerations for programs that access DB2 databases, and recovery and database integrity for DB2 programs running in the CICS environment.

Associating DB2 Databases with CICS Transactions

If the programs running under a transaction access DB2 databases, then you must create CICS DB2ENTRY or DB2TRAN resource definitions to associate the DB2 plan name with the CICS transaction code.

For information on the parameters you can specify when you create CICS DB2ENTRY or DB2TRAN resource definitions, refer to the CICS resource definition guide.

Recovery and Database Integrity Considerations

EGL-generated programs can use all the recovery and data integrity features that are provided by DB2 in the CICS environment.

Relational databases are recoverable resources. If your program makes changes to a relational database, the changes are not committed to the database until the end of a logical unit of work (LUW). If your program ends abnormally before the end of an LUW, all changes that were made since the beginning of the LUW are backed out. See "Specifying Recovery Options in CICS" on page 39 for more information about handling recovery in CICS. For information on when an LUW ends, refer to the EGL help topic "Logical unit of work."

Considerations for Using DL/I in CICS

This section discusses recovery and database integrity considerations for DL/I programs running in the CICS environment.

Refer to the EGL helps for additional information.

Recovery and Database Integrity Considerations

EGL-generated programs can make use of all the recovery and data integrity features that are provided by DL/I in the z/OS CICS environment.

DL/I databases are recoverable resources. If your program makes changes to a DL/I database, the changes are not committed to the database until the end of a logical unit of work (LUW). If your program ends abnormally before the end of an LUW, all changes that were made since the beginning of the LUW are backed out. See "Specifying Recovery Options in CICS" on page 39 for more information about handling recovery in CICS. For information on when an LUW ends, refer to the EGL help topic "Logical unit of work."

Setting up the National Language

On CICS, the national language code used for the first program in the run unit determines the language that is used for all messages for all programs in the run unit.

Chapter 7. System Considerations for z/OS Batch

This chapter presents system considerations for running EGL-generated programs in the z/OS batch environment.

The following information is discussed:

- Required file descriptions
- Using VSAM program data files
- Considerations for using DB2
- Considerations for using DL/I
- Performance considerations
- Runtime JCL

Required File Descriptions

Rational COBOL Runtime requires the following files:

File Name	Description
------------------	--------------------

EZEPRINT	This file is used when printing from a program that displays print forms. EZEPRINT can be allocated to either a data set or to a SYSOUT class. The file must have a VBA (variable-blocked ANSI) record format.
-----------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The maximum record length that a generated program can write to the print data set is 654 bytes for DBCS forms and 137 bytes for SBCS forms. The record length includes 4 bytes for the variable length record header, 1 byte for the American National Standards printer-control character, and the print line for the print form. The DBCS record length is longer than the print line length because the print line can contain outlining control characters and shift-out (SO) and shift-in (SI) characters that are not displayed on the device. The logical record length defined for the data set must be greater than or equal to the length of the longest line written by the program, including the DBCS SO/SI characters.

If you are using Rational COBOL Runtime to print to a file destination other than EZEPRINT, the characteristics of that file should be the same as EZEPRINT.

SYSPRINT, SYSOUT, SYSABOUT, SYSUDUMP	These z/OS system files are used by EGL-generated programs. Do not specify DCB parameters for these files.
---------------------------------------------	------------------------------------------------------------------------------------------------------------

ELAPRINT	This system output file is used by generated programs. Specify ELAPRINT with RECFM=FBA and BLKSIZE=1330 DCB parameters.
-----------------	-------------------------------------------------------------------------------------------------------------------------

ELATRACE	This file is the trace control file for the z/OS batch environment. The attributes for this data set are LRECL=80, RECFM=FB, and BLKSIZE=multiple of 80. The trace filters are specified in the ELATRACE data set.
-----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

ELATOUT	The output of the Rational COBOL Runtime trace facility is sent to this data set in the z/OS batch environment. The attributes for this data set are DSORG=PS, LRECL=133, BLKSIZE=1330, and RECFM=FBA.
----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Using VSAM Program Data Files in z/OS Batch

VSAM program data files must be defined before your z/OS batch program can use them. See “Defining and Loading VSAM Program Data Files” on page 28 for information on defining VSAM data sets, defining alternate indexes, and for information on loading VSAM data sets.

The DD statements for user files are generated for you and placed in the sample runtime JCL.

Considerations for Using DB2 in z/OS Batch

This section presents system considerations for database recovery and integrity for DB2 programs.

For information on running DB2 programs in z/OS batch, see Chapter 13, “Preparing and Running Generated Programs in z/OS Batch,” on page 101.

Recovery and Database Integrity Considerations

EGL-generated programs can use all the recovery and data integrity features provided by DB2.

Relational databases are recoverable resources. If your program makes changes to a relational database, the changes are not committed to the database until the end of a logical unit of work (LUW). If your program ends abnormally before the end of an LUW, all changes that were made since the beginning of the LUW are backed out. For information on when an LUW ends, see the EGL help topic “Logical unit of work.”

If a program runs in z/OS batch and accesses DB2, it can run in an RRSAP environment. The EGL help topic “Logical unit of work” also has details on enabling the RRSAP capability.

Considerations for Using DL/I in z/OS Batch

This section presents the following information:

- Defining the program specification block (PSB)
- Recovery and database integrity considerations

For information on running DL/I programs in z/OS batch, see Chapter 13, “Preparing and Running Generated Programs in z/OS Batch,” on page 101.

Defining the Program Specification Block (PSB)

The following list shows considerations for defining a PSB that is used in the z/OS batch environment:

- DL/I PSBs used in the z/OS batch environment must have CMPAT=YES specified in the PSBGEN statement for the PSB. This enables you to use the CHKP and ROLB functions with the PSB.
- The PSBGEN statement must include the parameter LANG=COBOL or LANG=ASSEM.
- DL/I PSBs used in the z/OS batch environment must be defined with a minimum of two PCBs of any type in the PSB. This enables the generated COBOL program to test whether it is being started from the IMS region

controller or from an OS XCTL macro in a non-EGL program passing working storage and **dliLib.psbData** as parameters.

- z/OS batch programs can implement serial files as GSAM databases. These GSAM files are treated as a special type of database and require a PCB in the PSB. The GSAM PCBs must follow all database PCBs.

Recovery and Database Integrity Considerations

In z/OS batch DL/I programs, a commit point causes a DL/I basic CHKP (checkpoint) call. The contents of **dliLib.psbData** are used as the checkpoint identifier. After the CHKP call, **dliVar.statusCode** contains the status code returned with the CHKP call.

If the program runs under the TSO terminal monitor program for SQL access, calling the **sysLib.rollback()** system function results in an SQL ROLLBACK WORK.

If the program runs as a DL/I batch job, and DL/I or SQL requests have been issued, calling the **sysLib.rollback()** system function results in a DL/I ROLB call. The IMS batch parameter BKO=Y must be specified when the batch job is started in order for the ROLB call to be honored. The BKO parameter is specified in the job step that calls the IMS control program DFSRRC00. If BKO=N is specified, DL/I returns status code AL for the ROLB call. Rational COBOL Runtime treats the AL as a soft error, and no error message is issued.

Serial or print files associated with GSAM files and the **sysLib.audit** system function result in DL/I requests and cause the DL/I ROLB call to be issued. For information on when a commit point or rollback is issued, refer to the EGL help topic "Logical unit of work."

Considerations for Calling CICS programs from z/OS batch

You must set up the CICS region to receive EXCI calls. For information, see "CICS Setup for Calling CICS Programs from z/OS Batch" on page 93.

Performance Considerations for z/OS Batch

See "Modules in Memory" on page 28 for information on performance considerations and the methods used to place modules in memory. These methods are particularly beneficial if the EGL program is being called repeatedly by a non-EGL program.

If you are running generated programs in z/OS batch and are accessing indexed or relative files, you do not need to use the **forUpdate** option on the I/O statement prior to a **delete** or **replace** statement. Eliminating the **forUpdate** option allows for better performance, as it eliminates a COBOL read. However, make sure that you perform a **get** or **get next** before the **delete** or **replace** to ensure that the record is available.

Runtime JCL

See Chapter 13, "Preparing and Running Generated Programs in z/OS Batch," on page 101 for examples of batch runtime JCL.

Chapter 8. System Considerations for IMS

This chapter provides additional administrative information that applies to the IMS environments.

The following information is discussed:

- Required file descriptions
- Defining the program specification block
- Processing modes
- Printing considerations for IMS
- Recovery and database integrity considerations
- Considerations that affect performance
- Considerations for using DB2
- Considerations for using DL/I
- Maintaining the work database
- Consideration for Message Format Services

Required File Descriptions

Rational COBOL Runtime requires the following files:

File Name	Description
ELASNAP	This is an optional file that contains the snap dump listing when a Rational COBOL Runtime error occurs and the ELASNAP DD statement was included in the startup JCL. This file has a 125-byte logical record length, a 882-record block size, and a VBA (variable-blocked ANSI) record format. If this file is directed to the SYSOUT system logical unit, define it with RECFM=VBA and BLKSIZE=4096.
ELAPRINT	This file is an optional output file for Rational COBOL Runtime error messages. This file has a fixed block record format, a 133-byte logical record length, and a block size of 1330. If this file is directed to the system logical unit SYSOUT, define it with RECFM=FBA and BLKSIZE=1330.
ELADIAG	This is the default name for the optional message queue for Rational COBOL Runtime error messages. This message queue is defined in the IMS system definition during Rational COBOL Runtime installation. See “IMS Diagnostic Message Print Utility” on page 135 for information about printing the error messages contained in the ELADIAG message queue.
ELATRACE	This is the trace control file for the IMS BMP environment. The attributes for this data set are LRECL=80, DSORG=PS, and BLKSIZE=multiple of 80. The trace filters are specified in the ELATRACE data set.
ELATOUT	The output of the Rational COBOL Runtime trace facility is sent to this data set in the IMS BMP environment. The attributes for this data set are LRECL=133, BLKSIZE=1330, and RECFM=FBA.
ELAT	The output of the Rational COBOL Runtime trace facility is sent to this output message queue in the IMS/VSE environment. Use the ELAMQJUD job to retrieve the trace.

EZEPRINT This is the default message queue (IMS/VS) or output file (IMS BMP) for print output from generated programs. For IMS BMP programs, the print records are variable length. For single-byte languages, define EZEPRINT with LRECL=137, BLKSIZE=141, and RECFM=VBA. For double-byte languages, define EZEPRINT with LRECL=654, BLKSIZE=658, and RECFM=VBA. If the file is directed to the system logical unit SYSOUT, define it with RECFM=VBA and BLKSIZE=4096.

Defining the Program Specification Block (PSB)

You need to define both an IMS PSB and an EGL PSB record for your program. The EGL PSB record contains a subset of the information from the IMS PSB and is used to build default segment search arguments (SSAs) for the EGL I/O statements.

You need to generate an IMS PSB to correspond to the EGL PSB record. For IMS/VS, the IMS PSB must have the same name as the load module for the associated EGL program. A program control block (ACB) generation is also required for the IMS/VS environment. For IMS BMP and DL/I batch, the IMS PSB name does not have to match the program load module name.

When you define the PSBs for IMS programs, consider the following criteria:

- The PSBGEN statement must include the parameters CMPAT=YES, and LANG=COBOL or LANG=ASSEM.
- The I/O PCB (program control block) is automatically supplied and does not appear in the IMS PSB. You must include the I/O PCB in the EGL PSB record if you specify the **callInterfaceType=CBLTDLI** property in your EGL program.
- Alternate PCBs are used to route output to terminals other than the originating terminal, or to other transactions. Alternate PCBs must appear before the database PCBs both in the IMS PSB and in the EGL PSB record.
- When an EGL program is generated for the IMS/VS or IMS BMP environment, a modifiable alternate PCB and a modifiable express alternate PCB are required, in that order, as the first two PCBs following the I/O PCB. Both of these PCBs must have the parameters ALTRESP=NO and SAMETRM=NO. To avoid having to edit your DL/I call modifications to adjust for the two required PCBs, include these PCBs whenever you plan to generate a program for the IMS/VS or IMS BMP target environments.
- IMS BMP programs can implement serial files as GSAM databases. These GSAM files are treated as a special type of database and require a PCB in both the IMS PSB and the EGL PSBRecord. The GSAM PCBs must follow all database PCBs.

If a DL/I work database is used, the PCB for this database must be included in the IMS PSB. This PCB can be created using the macro ELAPCB and concatenating ELA.V6R0M1;ELASAMP as part of the SYSLIB in the PSBGEN procedure. Figure 7 on page 53 shows an example of the PCB expansion that occurs when ELAPCB is used.

WORKDBD defaults to ELAWORK. The WORKDBD parameter must be used if the DBD name is changed.

```

ELAPCB   [WORKDBD=customer-dbd-name]

    --- expands into ---

PCB      TYPE=DB,DBDNAME=customer-dbd-name,PROCOPT=AP,KEYLEN=19
SENSEG   NAME=ELAWCNTL,PARENT=0
SENSEG   NAME=WORKLV01,PARENT=ELAWCNTL
SENSEG   NAME=WORKLV02,PARENT=WORKLV01
:
:
SENSEG   NAME=WORKLV14,PARENT=WORKLV13
SENSEG   NAME=MSGLV01,PARENT=ELAWCNTL
SENSEG   NAME=MSGLV02,PARENT=MSGLV01
:
:
SENSEG   NAME=MSGLV14,PARENT=MSGLV13

```

Figure 7. Generating the DL/I Work Database PCB

If you specify (or default to) the `callInterfaceType=DLICallInterfaceKind.AIBTDLI` property for your program, the EGL program refers to the PCBs in the PSB by name rather than by position. The default PCB names are as follows:

- IOPCB (required by IMS for the I/O PCB)
- ELAALT (the EGL default name for the modifiable alternate PCB)
- ELAEXP (the EGL default name for the modifiable express alternate PCB)
- ELAWORK (the EGL default name for the DL/I work database PCB).

Processing Modes

IMS requires segmented mode. Refer to the EGL help system for additional information on segmented mode.

The `spaSize="xxxx"` build descriptor option determines whether a program runs as IMS conversational (`xxxx` is greater than 0) or nonconversational (`xxxx` is 0). Refer to the *EGL Generation Guide* for more information.

The work database is used for both conversational and nonconversational processing to save information during a **converse**. In nonconversational mode, the work database is also used to save information during a deferred program-to-program message switch which results from a **show** statement. In conversational mode, the scratch-pad area (SPA) is used to set the transaction identifier and to save information during a program-to-program message switch. Refer to the *EGL Programmer's Guide* for information on how the SPA is used for program-to-program message switching.

Printing Considerations for IMS

From Rational COBOL Runtime, printing is initiated when a program processes a **print** statement for an EGL printForm. Refer to the EGL help system for information on defining forms for printers.

Printing is accomplished using MFS control blocks produced when the FormGroup is generated. The default print destination in IMS is a message queue named EZEPRINT. The printer destination can be changed at generation time. You can also change the print destination at run time by changing the **converseVar.printerAssociation**. Refer to the EGL help system for additional information.

Recovery and Database Integrity Considerations

EGL programs can make use of all the IMS recovery and data integrity features.

If your program makes changes to a recoverable resource, the changes are not committed until the end of a logical unit of work (LUW). If your program abnormally ends before the end of an LUW, all changes that were made since the beginning of the LUW are backed out. For information on when an LUW ends, see the EGL help topic "Logical unit of work."

Considerations that Affect Performance

This section describes factors that affect system performance and suggestions on how to improve performance.

Residency Considerations and the IMS Preload Function

The performance of a program is affected by the number of times a disk is accessed while running the program. Programs require access to disks for the following reasons:

- Locating and loading Rational COBOL Runtime load modules
- Retrieving and storing user data
- Locating and loading application, FormGroup format modules, MFS print services programs, and table load modules

Rational COBOL Runtime loads objects as they are needed. For example, Rational COBOL Runtime loads a program, MFS print services program, FormGroup format module, or DataTable when another program calls or references it. The overhead of locating and loading modules can be reduced by using the IMS preload function. Preloading an object reduces the amount of I/O required for multiple loads. However, preloading generated programs requires more virtual storage for your system because preloaded modules remain in storage until the message region is shut down.

It is usually not possible for everything to be preloaded. Therefore, you should establish priorities for deciding which objects you should preload. These preloading priorities reflect a trade-off between your program usage and your storage constraints. Because of individual considerations such as storage constraints, environment, and types of programs, your priorities might dictate that some components (such as MFS print services programs) for a program be preloaded, while other components (such as DataTables) should not be preloaded. Make the decision on what modules to preload on an individual basis, according to how the program uses them.

When deciding what to preload, consider the following:

- Storage constraints
- Frequency of program use
- Long-running programs as compared to programs that are started more frequently

Generally, objects that are loaded more than once are prime candidates for preloading. Examples of this are a DataTable that is used by more than one program and a program that is called more than one time. The following are some general rules for preloading:

- When deciding what to preload, consider the following objects:
 - Called programs
 - MFS print services programs

- FormGroup format modules
- DataTables
- Main programs
- Programs that are started or referenced frequently should be preloaded. In addition to programs that are loaded by IMS when a transaction is scheduled, this includes programs that are started by the EGL **transfer to program** or **call** statements.
- Programs that are not frequently initiated should not be preloaded.

See “Preloading Generated Programs” on page 56 for additional information.

Preloading Rational COBOL Runtime Modules

For best performance, use the preload option for the following Rational COBOL Runtime modules:

- ELARPRTR, the Rational COBOL Runtime module that handles address mode switching
- ELARPRTM, the Rational COBOL Runtime load module
- ELARPIOP, the installation options module
- ELARlccc (where *ccc* is the language code), the language-dependent options module
- ELACNccc (where *ccc* is the language code), the conversion table
- ELANCccc (where *ccc* is the language code), the module for Rational COBOL Runtime constants and the table that converts from lower case to upper case
- ELARSCNT, the configuration table
- ELA2SSQW, the module that supports the DB2 work database
- ELARSDCB, which is used for accessing Rational COBOL Runtime sequential files
- ELA2SSQL, its alias ELA2SSQY, and ELA2SSQX
 ELA2SSQL, its alias ELA2SSQY, and ELA2SSQX are used to gain access to the DB2 work database, and they support commit and rollback processing for DB2 program databases. Preload these modules only if you are using programs that were generated and bound using CSP/370RS V1R1.

The modules ELARSDCB and ELANCccc are loaded below the 16MB line. ELARSDCB is used only in reporting errors detected by Rational COBOL Runtime. Both can be omitted from the preload list if storage space below the 16MB line is limited.

Note: You should also monitor the usage of the LE runtime modules. Because many are used by the generated COBOL programs, these modules might also be candidates for preloading.

Refer to the IMS documentation for your system for information on the preload option. An alternative to preloading is to place modules in the link pack area.

Loading Rational COBOL Runtime Modules into the Link Pack Area

Placing modules in the link pack area causes all regions to share a single copy of the modules and saves storage space. Refer to the Rational COBOL Runtime program directory for information about what modules can be put into the link pack area.

Only one version of CSP/370RS V2R1, CSP/370RS V1R1, VisualAge Generator Server V1R2, Enterprise Developer Server or IBM Rational COBOL Runtime modules can be placed in the link pack area. If multiple releases are installed concurrently on the same system, override the link pack area by defining the correct library in the STEPLIB or JOBLIB DD statements for the region.

Preloading Generated Programs

You can reduce the overhead of searching the STEPLIB, JOBLIB, link pack area, and link list by preloading generated programs (application programs, online print services programs, FormGroup format modules, and DataTable modules) that are frequently used. However, in this case, virtual storage is still occupied by the modules when they are not in use.

To improve response time, you might also preload any module associated with any transaction that might require better performance, even though the module itself is not frequently used.

To preload generated programs, do the following:

1. Put the module in a LNKST library.
2. Include the module name in a preload member (DFSMPxxx, where xx is a two-character ID that you select) in the IMS procedure library.
3. Indicate in the JCL for the IMS message region that the preload member is to be included.

Database Performance

Database performance can be improved under IMS/ESA by defining HIPERSPACE* buffer usage for IMS in the DFSVSMxx member. This is the same as defining many buffers for the files, but has the advantage that the HIPERSPACE buffers all come from 31-bit storage, not from within the IMS/ESA region. The tuning of database buffer pools is recommended. Refer to the IMS manuals for details on the tuning of database buffer pools.

If you have IMS/ESA installed and use a DL/I work database, make the work database nonrecoverable to reduce the amount of logging that occurs. Making the work database nonrecoverable might help improve performance.

Limiting MFS Control Blocks

Limiting the size and number of message format service (MFS) control blocks might help improve performance. MFS is used for form support in the IMS environment. MFS control blocks are generated using MFS utility control statements.

You can reduce the size and number of MFS control blocks that are generated by doing the following:

- In form definition, only include the **screenSizes** values that are used for the application system. For additional information about the valid **screenSizes** values, refer to the EGL help system.
- Include in the **mfsDevice** build descriptor option only the combinations of the **height**, **width**, and **devStmtParms** properties that your installation or application system uses. For additional information about specifying the **mfsDevice** build descriptor option, refer to the *EGL Generation Guide*.

Monitoring and Tuning the IMS System

You can track potential performance problems before they occur by checking processing statistics on a regular basis. The following are some of the statistics to monitor:

- Use the IMS monitor facilities to check transaction utilization. Consider preloading programs or groups of programs which are frequently used.
- Use the IMS database monitor facilities to check how effectively the databases are performing and using space.

You can also use the following tools to monitor IMS performance:

- IMS Performance Monitor for z/OS (program number 5655-G50). This tool provides real-time status monitoring and alerts for IMS subsystems, as well as access to recent historical data and detailed statistical reports.
- IMS Performance Analyzer for z/OS (program number 5655-R03). This tool provides comprehensive performance analysis and tuning assistance for IMS, including end-to-end transit analysis for transaction workloads and availability of important resources such as databases and message queues.

Refer to the system administration manuals and the database administration guide for your release of IMS for detailed information about monitoring the IMS online system and DL/I databases.

Considerations for Using DB2 in IMS

This section discusses considerations for recovery, database integrity, and security issues for DB2 programs.

For information on designing and generating DB2 programs for the IMS environment, refer to the EGL help system.

For information on preparing DB2 programs for running in the IMS environment, see Chapter 14, "Preparing and Running Generated Programs in IMS/VS and IMS BMP," on page 107.

Recovery and Database Integrity Considerations

EGL-generated programs can use all the recovery and data integrity features that are provided by DB2 in the IMS environment.

Relational databases are recoverable resources. If your program makes changes to a relational database, the changes are not committed to the database until the end of a logical unit of work (LUW). If your program ends abnormally before the end of an LUW, all changes that were made since the beginning of the LUW are backed out. For information on when an LUW ends, see the EGL help topic "Logical unit of work."

Checking Authorization

The database manager checks whether the program users have authority to gain access to tables or to run programs. The type of checking done varies depending on your system and the processing mode.

When using DB2 in generated COBOL programs, the program users must be authorized to run the corresponding DB2 plan. For transaction-oriented regions, the authorization ID depends on the type of IMS security being used:

- If sign-on security is used, IMS provides the sign-on name as the authorization ID.
- If sign-on security is not used, IMS provides the name of the originating terminal as the authorization ID.

The DB2 plan used with a transaction has the same name as the program associated with the transaction.

For batch-oriented regions, the authorization ID is the contents of the ASXBUSER field, if valid, or the PSB name. The DB2 plan name is specified as one of the batch program parameters.

For more information on IMS security mechanisms, refer to the appropriate IMS manual.

Considerations for Using DL/I in IMS

This section discusses considerations for DL/I programs in the IMS environment.

See “Defining the Program Specification Block (PSB)” on page 52 for information on defining a PSB for DL/I programs.

For information on designing and generating DL/I programs for the IMS environment, refer to the EGL help system.

For information on preparing DL/I programs for running in the IMS environment, see Chapter 14, “Preparing and Running Generated Programs in IMS/VS and IMS BMP.”

Recovery and Database Integrity Considerations

EGL-generated programs can make use of all the recovery and data integrity features that are provided for DL/I databases in the IMS environment.

DL/I databases are recoverable resources. If your program makes changes to a DL/I database, the changes are not committed to the database until the end of a logical unit of work (LUW). If your program ends abnormally before the end of an LUW, all changes that were made since the beginning of the LUW are backed out. For information on when an LUW ends, see the EGL help topic “Logical unit of work.”

Maintaining the Work Database in IMS

You should monitor and tune the DL/I and DB2 work databases just as you would any other DL/I database or DB2 table. You can use the normal database administration utilities to monitor these databases and to determine when they need to be reorganized to improve performance.

The activities involved in maintaining the work database are the following:

- Deleting old records from the work database
- Expanding the work database
- Supporting multiple DL/I or DB2 work databases

Deleting Old Records from the Work Database

The terminal ID is the key for the records in the work database. Each record contains a time stamp that indicates the last time the record was updated.

Deleting old records from the database reduces the amount of disk space required in the work database. You probably want to delete records in the following situations:

- Some users might run a generated program only infrequently, less than once a day, for example. In this case, you might want to delete old records on a daily or weekly basis.
- Sometimes terminal names are changed or users are moved to terminals with different names. In this case, new records are created for the new terminals, but the old records are not automatically deleted.

The utilities that delete records from the DL/I and DB2 work databases validate the date and time to ensure that your request does not result in deletion of records that are less than 24 hours old.

DL/I Work Database

Figure 8 shows the JCL used to remove old records from a DL/I work database. The JCL is supplied as member ELAWKJCD in the ELA.V6R0M1;ELAJCL file. Specify the records you want to delete by entering the date (in Julian format) and time prior to which all records are to be deleted.

```

//*****
//** ELAWKJCD - JOBSTREAM TO CLEAN UP THE DLI WORK DATABASE
//**          FOR IBM RATIONAL COBOL RUNTIME.
//**
//** LICENSED MATERIALS - PROPERTY OF IBM
//** 5655-R29 (C) COPYRIGHT IBM CORP. 1994, 2006
//** SEE COPYRIGHT INSTRUCTIONS
//**
//** STATUS = VERSION 6, RELEASE 0, LEVEL 1
//**
//** TO TAILOR THIS JOBSTREAM:
//**     1. COPY A JOBCARD.
//**     2. REPLACE DATE AND TIME STAMP VALUE WITH DESIRED
//**        VALUE. ALL RECORDS WITH LESS THAN THAT DATE AND
//**        TIME WILL BE DELETED.
//**
//** RETURN CODES
//**  0 - SUCCESSFUL COMPLETION
//** 12 - FATAL ERROR. INVALID INPUT
//** 16 - FATAL ERROR. PROCESSING TERMINATED
//**
//*****
//*
//DLIWORK    EXEC IMSBATCH,MBR=ELAWKPC1,
//           PSB=ELAWKPB1,RGN=4096K
//G.STEPLIB  DD
//           DD
//           DD DSN=CEE.SCEERUN,DISP=SHR
//           DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
//G.ELAPRINT DD SYSOUT=*
//G.SYSOUT   DD SYSOUT=*
//G.SYSIN    DD *
YYDDHHMMSS

```

Figure 8. JCL to Remove Old Records from DL/I Work Database

DB2 Work Database

Figure 9 on page 60 shows the JCL used to remove old records from a DB2 work database. The JCL is supplied as member ELAWKJC2 in the ELA.V6R0M1;ELAJCL file. Specify the records you want to delete by entering the date (in Julian format)

and time prior to which all records are to be deleted.

```

/*****
/** ELAWKJC2 - JOBSTREAM TO CLEAN UP THE DB2 WORK DATABASE
/**          FOR IBM RATIONAL COBOL RUNTIME.
/**
/** LICENSED MATERIALS - PROPERTY OF IBM
/** 5655-R29 (C) COPYRIGHT IBM CORP. 1994, 2006
/** SEE COPYRIGHT INSTRUCTIONS
/**
/** STATUS = VERSION 6, RELEASE 0, LEVEL 1
/**
/** TO TAILOR THIS JOBSTREAM:
/**          1. COPY A JOBCARD.
/**          2. REPLACE DATE AND TIME STAMP WITH THE DESIRED DATA.
/**          ALL ROWS WITH A DATE AND TIME LESS THAN THE
/**          SPECIFIED DATE/TIME WILL BE DELETED.
/**
/** RETURN CODES
/**          0 - SUCCESSFUL COMPLETION
/**          12 - FATAL ERROR. INVALID INPUT
/**          16 - FATAL ERROR. PROCESSING TERMINATED
/**
/*****
/*
//DB2WORK EXEC PGM=ELAWKPC2,REGION=4096K
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//ELAPRINT DD SYSOUT=*
//ELASNAP DD SYSOUT=*
//EZESPUFI DD DSN=&&TMP1,DISP=(NEW,PASS),
//          UNIT=SYSDA,SPACE=(TRK,(1,0)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//SYSIN DD *
YYDDHHMMSS
/*
//DB2SPUF EXEC PGM=IKJEFT01,REGION=4096K,COND=(0,NE)
//STEPLIB DD DSN=DSN.RUNLIB.LOAD,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD DSN=&&TMP1,UNIT=SYSDA,DISP=(OLD,DELETE)
/*
//SYSTSIN DD *
DSN SYSTEM(DSN)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA??)
END
/*

```

Figure 9. JCL to Remove Old Records from DB2 Work Database

Expanding the Work Database

At times, you need to expand the work database. For example, you need to expand the database when you expand the usage of an existing program system to a larger user set comprising a much larger number of terminals that gain access to EGL-generated programs.

DL/I Work Database

To expand the DL/I work database, perform the following steps:

1. Stop the DL/I database.
2. Unload the database using the old database description (DBD).

3. Change the DBD information and perform a DBD generation.
4. If you are having application control blocks (ACBs) prebuilt rather than built dynamically, build the ACBs again.
5. Delete the space allocated for the old database and allocate space for the new definition.
6. Load the database using the new DBD.
7. Make an image copy of the new database for back-up purposes as soon as it is loaded.

Refer to the database administrator's guide and the IMS utilities manual for additional information.

DB2 Work Database

You might need to expand the table spaces containing the DB2 work database because of degraded performance from too many secondary extents, or because the application users receive a DB2 message DSNP007I indicating that no more space is available.

Ideally, when the size of a DB2 table space is increased, the primary extent should be made large enough to accommodate all the data in the work database. In any case, try to minimize the number of secondary extents required to store rows in the database.

The method you use to expand the table space depends on the version of DB2 that is installed and whether the table space is user-managed.

The procedure supplied with Rational COBOL Runtime that installs the work database also installs the table space as user-managed table space (no associated DB2 storage group).

Before attempting to change the size of the table space data set, you need to estimate the space requirements for the table space. One factor in your estimate is the amount of space currently used. If the space is currently DB2-managed (resulting from an earlier change in space allocation), you can get this information by first running the DB2 STOSPACE utility against the table space storage group, and then running the following query:

```
SELECT SPACE
  FROM SYSIBM.SYSTABLEPART
 WHERE TSNAME='tsname' and DBNAME='dbname';
```

The result (SPACE) gives the number of kilobytes of storage currently allocated to the table space.

If the space for the table space is user-managed, you can use the TSO LISTCAT command to obtain the space information. You need to know the data set name of the VSAM file used for table space. The data set name for the VSAM file has the following format:

```
catname.DSNDBC.dbname.tsname.I0001.Annn
```

where:

catname Specifies the VSAM catalog name or alias

This is the same name or alias as in the USING VCAT clause of the CREATE TABLESPACE statement.

dbname Specifies the DB2 database name

This is the same as the database name in the CREATE TABLESPACE statement.

tsname

Specifies the table space name

This is the same as the table space name in the CREATE TABLESPACE statement.

nnn

Specifies the data set number

For partitioned table spaces, the number is 001 for the first partition, 002 for the second, and so forth, up to the maximum of 64 partitions. For a simple or segmented table space, the number is 001 for the first data set. If the simple or segmented table space exceeds 2 gigabytes, the second data set is 002, and so forth.

To expand table space do the following:

1. Stop the DB2 database by using the command `-STOP DB (dbname)`.
2. Make an image copy of the table space. You can use the image copy to restore the data set if the procedure is not successful.
3. Create a storage group for the table space. Do this only if the table space currently is user-managed and a storage group is not already available.
4. Change the table space definition as follows:

- If the table space data sets are user-managed, use a DB2 statement as follows:

```
ALTER TABLESPACE dbname.tsname
  USING STOGROUP stogrp
  PRIQTY pppp SECQTY ssss
```

where:

dbname.tsname	Specifies the name of the space
stogrp	Specifies the name of the storage group
pppp	Specifies new primary allocation size (in kilobytes) for the expanded table space
ssss	Specifies new secondary allocation size (in kilobytes) for the expanded table space

Note: This statement changes the table space from user-managed to DB2-managed.

- If the table space data sets are already DB2-managed, use a DB2 statement as follows:

```
ALTER TABLESPACE dbname.tsname
  PRIQTY pppp SECQTY ssss
```

where:

dbname.tsname	Specifies the name of the space
pppp	Specifies new primary allocation size (in kilobytes) for the expanded table space
ssss	Specifies new secondary allocation size (in kilobytes) for the expanded table space

5. Move the table space data. Simply changing the table space definition does not put the new size into effect. You need to move the table space to the newly allocated space. You can, for example, reorganize the table space using the DB2 REORG utility.

6. Start the DB2 database. Enter the command `-START DB (dbname)`.

Supporting Multiple Work Databases

You can use separate work databases for different application systems. For example, you might want to use separate databases for payroll and shipping to improve performance or to increase data availability. The work database is used to pass information during certain types of program-to-program message switches between applications. When this occurs, both the transferring application and the transferred-to application must use the same physical work database.

DL/I Work Databases

To create an additional DL/I work database called ELAWORK2, do the following:

1. Copy the ELAWORK DBD in the ELA.V6R0M1;ELASAMP file, and name it ELAWORK2.
2. Change the NAME parameter on the DBD statement to ELAWORK2. Also change the DD1 parameter on the DATASET statement to ELAWORK2. Make any other changes to the block size, number of blocks, and randomizing routine based on the application system requirements.
3. Make copies of the ELAWKLD and ELAWKPB1 program specification blocks (PSBs) in the ELA.V6R0M1;ELASAMP file and give them new member names. Change the NAME parameter on the program control block (PCB) statement from ELAWORK to ELAWORK2.
4. Modify job ELACJWKD in the ELA.V6R0M1;ELAJCL file to refer to the new database. This job does the DBD, PSB, and ACB generations needed for the work database, allocates the database, and then initializes it. You need to change the DD and data set names for the work database, and name the new DBD and PSB.
5. Add the new database to the JCL for your IMS control region, and to your IMS stage-1 system definition.
6. When you create IMS PSBs for applications that need to use this new database, use the ELAPCB macro to create the PCB definition for the work database. Enter the following command:

```
ELAPCB WORKDBD=ELAWORK2
```
7. If you specify (or default to) the **callInterfaceType=DLICallInterfaceKind.AIBTDLI** property for your program, specify the **PCBName** property for the ELAWORK database in your EGL PSB record as follows:

```
ELAWORK DB_PCBRecord {@PCB {pcbType = PBKind.DB, PCBName = "ELAWORK2"}};
```

DB2 Work Databases

To create an additional DB2 work database, do the following:

1. Create an ELAWORK table using the ELACJWK2 job in the ELA.V6R0M1;ELAJCL file. Perform the following steps before running the job:
 - a. Add an authorization ID to the CREATE TABLE command in ELAWORK2 in the ELA.V6R0M1;ELASAMP file, for example:

```
CREATE PAYROLL.ELAWORK
```
 - b. Change the table space name and index in ELAWORK2.
 - c. Change the DELETE and DEFINE CLUSTER statements to use the table space name and index you specified in ELAWORK2.
 - d. Comment out the WRKDROP step to avoid dropping the existing work database.

- Each developer or system administrator using the payroll ELAWORK table needs to create a SYNONYM for the table. The following example shows how to use the CREATE SYNONYM command to create a synonym:

```
CREATE SYNONYM ELAWORK FOR PAYROLL.ELAWORK
```

The default BIND commands generated by EGL bind DBRMs for Rational COBOL Runtime modules to the program being generated. The CREATE SYNONYM command ensures that developers referencing the ELAWORK table use the payroll version of the table.

Considerations for Message Format Services in IMS

EGL generates message format services (MFS) source statements used for conversing and printing forms in IMS environments. The generated MFS source includes DEV statements, which identify the device types on which forms can be displayed and the characteristics of those devices. The device types and characteristics must be compatible with the device types and characteristics defined in the TERMINAL and TYPE macros in your IMS system definition.

The information on the generated MFS DEV statements is controlled by the **mfsExtendedAttr**, **mfsIgnore**, and **mfsDevice** build descriptor options. Review your TERMINAL and TYPE definitions and then set the **mfsExtendedAttr**, **mfsIgnore**, and **mfsDevice** build descriptor options to reflect your IMS system definition.

The following build descriptor options affect the generated MFS source:

mfsExtendedAttr

Specifies whether EGL generation includes extended attributes for the MFS DFLD statements if the information for the device size is not completely specified in the **mfsDevice** build descriptor option. The following values are valid:

- NO** NO specifies that extended attributes are not to be used. Specify NO if most of your devices do not support color or extended highlighting. NO specifies that EGL generation should omit the EATTR parameter from the MFS DFLD statements unless overridden by the **mfsDevice** build descriptor option for a specific device.
- YES** YES specifies that you want the default handling for extended attributes on the MFS DFLD statement. Specify YES if all of your devices support extended attributes (for example, devices that support color or extended highlighting), and you want EGL generation to include the CD (color default) extended attribute value when generating a form field that is defined with **color = mono** (monochromatic). YES specifies that EGL generation should include the EATTR parameter for MFS DFLD statements unless overridden by the **mfsDevice** build descriptor option for a specific device. YES is the default value.
- NCD** NCD specifies that EGL generation should include the EATTR parameter, but not include the CD extended attribute value for the MFS DFLD statements when generating a form field that is defined with **color = mono**.

The **mfsExtendedAttr** build descriptor option specifies how the DFLD statements for a specific device are to be generated if the EATTR, NCD, or NOEATTR parameter is not included in the **mfsDevice** build descriptor

option for a particular device size. If EATTR, NCD, or NOEATTR is specified for a particular device size in the **mfsDevice** build descriptor option, the **mfsExtendedAttr** build descriptor option has no effect for that device size.

mfsIgnore

Specifies the information EGL generation includes for the MFS MSG statement for the message input descriptor (MID) and message output descriptor (MOD). The following values are valid:

- YES** Specifies that you want EGL generation to include SOR= (... , IGNORE) on the MFS MSG statement for the MID and the MOD. Specify YES only if the **mfsDevice** option specifies FEAT=IGNORE for all the devices used by the FormGroup you are generating.
- NO** Specifies that you do not want EGL generation to include the SOR parameter on the MFS MSG statement for the MID and the MOD. The default is NO.

mfsDevice

Specifies the information that EGL generation uses for the MFS DEV and DFLD statements. This build descriptor option provides the correspondence between the EGL device size information that a developer specifies for a form and the device information that must be included for the MFS DEV statements.

To specify the **mfsDevice** build descriptor option, edit your build descriptor part using the EGL Build Parts Editor. In the upper right corner of the EGL Build Parts Editor window, click the **Show MFS Devices Properties** icon. The MFS Devices Properties editor appears. You can enter the following information:

Height

The number of lines that can be displayed on the device (for example, 24). This attribute is required.

Width The number of columns that can be displayed on the device (for example, 80). This attribute is required.

Device Statement Parameters

A string that contains one or more parameters you want EGL to include when generating the MFS DEV statement. Base this information on the TERMINAL and TYPE macros in your IMS system definition. This attribute is required.

Extended Attributes

Indicates whether the device supports extended attributes and whether a color default (CD) extended attribute is generated for form fields that are displayed on monochromatic devices. Your choice affects the EGL-generated MFS DFLD statements. If you specify this attribute, the value of build descriptor option **mfsExtendedAttr** is ignored when you generate form information for the device. Valid values are as follows:

YES (the default)

Extended attributes are supported, and a color default extended attribute is generated.

NCD Extended attributes are supported, but a color default extended attribute is not generated.

NO Extended attributes are not supported.

Note:

- The combination of **Height** and **Width** must match the values for the **screenSizes** property that developers specify for textForms and the values for the **formSize** property that developers specify for printForms.
- You can repeat the combination of **Height** and **Width** as many times as necessary to provide the correspondence to all your physical devices that match that device size. For example, if for **screenSize** =[24,80] for a textForm, you use both a 3270-A2 and a 3270-A3, you should include two entries for **Height**=80, **Width**=24, one for each device that you use.
- Include entries only for physical devices that you actually use. Including devices that you do not use increases the MFS control block size and can degrade performance.

If you do not specify the **mfsDevice** build descriptor option, the default value is shown in the following table.

Table 9. Default values for mfsDevice build descriptor option

Height	Width	Device Statement Parameters	Extended Attributes
80	24	TYPE=3270-A2,FEAT=(IGNORE)	YES
80	24	TYPE=(3270-2),FEAT=(IGNORE)	YES
132	255	TYPE=3270P,WIDTH=133,PAGE=(255,DEFN),FEAT=2	YES

The following table shows the relationship between the **mfsIgnore** and **mfsDevice** build descriptor options and the FEAT parameter for the TERMINAL and TYPE macros in the IMS system definitions.

Table 10. Relationship between mfsIgnore, mfsDevice, and the IMS System Definition

mfsIgnore	MFS MSG Statement for MID / MOD	mfsDevice FEAT Parameter	IMS System Definition FEAT Parameter
YES	SOR=(xxxx,IGNORE)	FEAT=IGNORE (see Note 1)	FEAT=IGNORE or FEAT= <i>n</i>
YES(2)		FEAT= <i>n</i> (see Note 2)	
	SOR=xxxx	FEAT=IGNORE (see Note 3)	FEAT=IGNORE
	SOR=xxxx	FEAT= <i>n</i> (see Note 3)	FEAT= <i>n</i>

Note:

1. The value for FEAT in the **mfsDevice** build descriptor option does not need to match the value for FEAT in the IMS TERMINAL or TYPE macro.
2. This combination of the **mfsIgnore** and **mfsDevice** build descriptor options is not valid. Generation ignores any device that uses this combination because the combination is not supported by MFS.
3. The value for FEAT in the **mfsDevice** build descriptor option must exactly match the value specified for FEAT in the IMS TERMINAL or TYPE macro.

The following table shows parameters from the **TERMINAL** and **TYPE** macros in your IMS system definition that you can code for the **Device Statement Parameters** in the **mfsDevice** build descriptor option. Do not code other MFS parameters for the MFS DEV statement in the **mfsDevice** build descriptor option.

Table 11.

Description	Device Statement Parameters	Optional Device Statement Parameters
3270 Display or 5550 Display	(3270,1), (3270,2), 3270-An (see Note 1)	FEAT
3270 Printer	3270P	FEAT, WIDTH (see Note 2), PAGE (see Note 3)
SCS1 Printer or 5550P Printer	SCS1	FEAT, WIDTH (see Note 4), PAGE (see Note 3)

Note:

1. The n in 3270-An is any number from 1 through 15.
2. If WIDTH is coded, FEAT must be coded. WIDTH must be a value 1 greater than the width for the **Width** attribute for the device size because the last column is used by MFS for carriage control. To have compatibility for a 3270 printer, use FEAT=n (where n is a value from 1 through 10 and matches your IMS system definition), WIDTH=133, PAGE=(255,DEFN).
3. If PAGE is coded and the second parameter is given, it must be DEFN. DEFN is the default.
4. To have compatibility for a SCS1 printer, use the following settings:
 - For a single-byte printer, use WIDTH=132, PAGE=(255,DEFN).
 - For a double-byte printer (such as a 5550P), use WIDTH=158, PAGE=(255,DEFN).

For assistance in setting the values for the **mfsExtendedAttr**, **mfsIgnore**, and **mfsDevice** build descriptor options, refer to the IMS system definition reference manual for your release of IMS for additional information on the parameters for the **TERMINAL** and **TYPE** macros. Also refer to the stage 1 system definition macros for your IMS system to determine the parameters actually used for your installation. Refer to the MFS manuals for your release of IMS for additional information about the DEV statement.

If you have IMS systems that are not generated from EGL, you might also want to look at some MFS source from those systems to see the parameters that you specify on the MFS DEV statement.

Once you have determined the correct values for the **mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore** build descriptor options, code the default build descriptor options in all the default build descriptor files that you use when generating for the IMS/VS or IMS BMP target environments.

The following table lists some example values that you might want to use for the **mfsDevice** build descriptor option.

Table 12. Example values for mfsDevice build descriptor option

Height	Width	Device Statement Parameters	Extended Attributes
80	24	TYPE=3270-A2,FEAT=(IGNORE)	YES
80	24	TYPE=(3270-2),FEAT=(IGNORE)	YES
80	24	TYPE=3270-A3,FEAT=(IGNORE)	YES
80	43	TYPE=3270-A4,FEAT=(IGNORE)	YES
132	27	TYPE=3270-A7,FEAT=(IGNORE)	YES
132	255	TYPE=3270P,WIDTH=133,PAGE=(255,DEFN),FEAT=2	YES
132	255	TYPE=SCS1,WIDTH=132,PAGE=(255,DEFN)	YES
132	255	TYPE=SCS1,WIDTH=158,PAGE=(255,DEFN)	YES

Part 3. Preparing and Running Generated Applications

Chapter 9. Output of Program Generation on z/OS Systems	71
Allocating Preparation Data Sets	71
List of Program Preparation Steps after Program Generation	73
Deploying generated code to USS	74
Output of Generation	74
Objects Generated for Programs	77
Application COBOL Program	77
Sample Runtime JCL	77
Bind Commands.	78
Link Edit File.	78
CICS Entries	78
Objects Generated for DataTables	78
DataTable COBOL Program	78
Objects Generated for FormGroups	79
Online Print Services Program	79
Batch Print Services Program	79
FormGroup Format Module	79
MFS Print Services Program	79
MFS Source	79
COBOL Copybook for MFS MID/MOD Layout	79
Chapter 10. z/OS Builds.	81
z/OS Build Server	82
Starting a z/OS Build Server	83
Starting a USS Build Server	85
Stopping servers.	85
Configuring a build server	85
Working with Build Scripts	85
Working with z/OS Build Scripts	85
Writing a JCL build script	87
File Name Conversions for z/OS	87
Converting JCL to Pseudo-JCL	87
Chapter 11. Preparing and Running a Generated Program in CICS	91
Modifying CICS Resource Definitions	91
Program Entries	91
Transaction Entries	92
Destination Control Table Entries	92
File Control Table Entries.	93
DB2 Entries	93
Using Remote Programs, Transactions, or Files	93
CICS Setup for Calling CICS Programs from z/OS Batch	93
CICS Setup for Calling z/OS Batch Programs in CICS	93
Modifying CICS Startup JCL.	94
Making New Modules Available in the CICS Environment	94
Making Programs Resident	95
Running Programs under CICS.	95
Starting the Transaction in CICS	95
Controlling Diagnostic Information in the CICS Environment	95
Printing Diagnostic Messages in the CICS Environment	95
Chapter 12. Creating or Modifying Runtime JCL on z/OS Systems	97
Tailoring JCL before Generation	97
Modifying Runtime JCL	98
Chapter 13. Preparing and Running Generated Programs in z/OS Batch	101
Running Main Programs under z/OS Batch	101
Examples of Runtime JCL for z/OS Batch Programs.	101
Running a Main Basic Program with No Database Access	102
Running a Main Basic Program with DB2 Access.	102
Running Main Basic Program with DL/I Access	103
Running a Main Basic Program with DB2 and DL/I Access.	104
Recovery and Restart for z/OS Batch Programs	105
Chapter 14. Preparing and Running Generated Programs in IMS/VS and IMS BMP	107
Modifying the IMS System Definition Parameters	107
Defining an Interactive Program	107
Defining Parameters for a Main Basic Program as an MPP	108
Defining Parameters for a Batch-Oriented BMP Program	109
Defining Parameters for a Transaction-Oriented BMP Program	109
Creating MFS Control Blocks	109
Making New Modules Available in the IMS Environment	110
Preloading Program, Print Services, and DataTable Modules	110
Running Programs under IMS.	111
Starting a Main Program Directly.	111
Starting a Main Transaction Program Using the /FORMAT Command	111
Running Transaction Programs as IMS MPPs	111
IMS Commands	111
Keyboard Key Operation	112
DBCS Data on a Non-DBCS Terminal	112
Error Reporting.	112
Responding to IMS Error Messages	112
Running Main Basic Programs as MPPs	113
Running a Main Basic Program under IMS BMP	113
Examples of Runtime JCL for IMS BMP Programs	114
Running a Main Basic Program as an IMS BMP Program	114

Running a Main Basic Program as an IMS BMP
Program with DB2 Access 115
Recovery and Restart for IMS BMP Programs. . . 116

**Chapter 15. Moving Prepared Programs to
Other Systems from z/OS Systems 117**
Moving Prepared Programs To Another z/OS
System 117
Maintaining Backup Copies of Production Libraries 118

Chapter 9. Output of Program Generation on z/OS Systems

This chapter provides an overview of the files produced at generation time and of the steps needed to prepare code for use at run time.

Output files are transferred to z/OS, where preparation steps include running translators, precompilers, and compilers; doing link-edits; and defining control tables for the target runtime environment.

For additional information on the output of program generation, refer to the *EGL Generation Guide* in the online help.

Allocating Preparation Data Sets

EGL COBOL generation creates and runs a build plan file. The build plan file controls the transfer of generated objects to the z/OS host and the execution of build scripts that are used to prepare the other output of generation.

The transferred objects are stored in partitioned data sets. You allocate the required data sets using the ELACUSER CLIST shipped in the Rational COBOL Runtime data set that has the low-level qualifier ELACLST. This CLIST was customized at product installation to set keyword default values to settings appropriate for your environment.

For you to use this CLIST, your customized data set **must** be placed before the Rational COBOL Runtime data set that has the low-level qualifier SELACLST in the SYSPROC concatenation list. Make sure that every COBOL generation user has the required data sets allocated for **every** target runtime environment in which the product will be used.

The following keyword parameters within CLIST ELACUSER may either be customized within the CLIST or overridden when executing the CLIST:

Keyword	Possible Values
ZOSBATCH	<ul style="list-style-type: none">• Y = allocate user data sets for this environment• N = do not allocate user data sets for this environment
ZOSCICS	<ul style="list-style-type: none">• Y = allocate user data sets for this environment• N = do not allocate user data sets for this environment
IMSBMP	<ul style="list-style-type: none">• Y = allocate user data sets for this environment• N = do not allocate user data sets for this environment
IMSVS	<ul style="list-style-type: none">• Y = allocate user data sets for this environment• N = do not allocate user data sets for this environment
VOL	vvvvvv = serial number
UNIT	uuuuuu = valid unit name

HLQ hhhhhhhh = high-level qualifier for user data sets

CLST

- FB = allocate a fixed blocked CLIST library
- VB = allocate a variable blocked CLIST library

DB2

- Y = DB2 databases will be used with this product
- N = DB2 databases will not be used with this product

CBLK

cccccc = CLIST data set block size

LBLK

llllll = load library data set block size

An example of the command syntax to execute the CLIST is as follows:

```
ex 'myRuntime.v5r0m0.elaclst(elacuser) zoscics(y) zosbatch(y)
vol(at1235) unit(sysda) hlq(tsouid) db2(y)'
```

Table 13 describes the data sets that are allocated. The DD name in the table is the DD name in the build scripts that are used by the build server. The meaning of lower-case strings in the data set name is as follows:

cghlq The high-level qualifier specified for the HLQ parameter in the ELACUSER CLIST.

env The generation environment. One of these:

- ZOSBATCH (for z/OS batch)
- ZOSCICS (for z/OS CICS)
- IMSVS (for IMS/VS)
- IMSBMP (for IMS BMP)

Table 13. Program Preparation User Data Set Information

DD Name	Data Set Name	Description	DCB Information	Target Environment
DBRMLIB	cghlq.env.DBRMLIB	Database request module library for DB2 programs	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	All z/OS, if DB2 used
EZEBIND	cghlq.env.EZEBIND	Bind commands	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	All z/OS, if DB2 used
EZECOPY	cghlq.env.EZECOPY	Generated message input descriptor (MID) and message output descriptor (MOD) layout copybooks.	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	IMSVS, IMSBMP
EZEFOBJ	cghlq.env.EZEFOBJ	Form group format object modules	DSORG=PO, RECFM=FB, BLKSIZE=3120, LRECL=80	ZOSCICS, IMSVS, IMSBMP
EZEJCLX	cghlq.env.EZEJCLX	Basic program runtime job stream	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	ZOSBATCH, IMSBMP
EZELINK	cghlq.env.EZELINK	Generated link edit control file	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	All z/OS
EZEMFS	cghlq.env.EZEMFS	Generated message format services control block source	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	IMSVS, IMSBMP

Table 13. Program Preparation User Data Set Information (continued)

DD Name	Data Set Name	Description	DCB Information	Target Environment
EZE OBJ	cghlq.env.OBJE CT	Object library	DSORG=PO, RECFM=U, BLKSIZE=6144, LRECL=0	All z/OS
EZEPCT	cghlq.env.EZEPCT	CICS PCT entries or RDO TRANSACTION entries	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	ZOSCICS
EZEPPT	cghlq.env.EZEPPT	CICS PPT entries or RDO PROGRAM entries	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	ZOSCICS
EZESRC	cghlq.env.EZESRC	COBOL source library for generated programs, libraries, or services	DSORG=PO, RECFM=FB, BLKSIZE=6160, LRECL=80	All z/OS
SYSLIN	cghlq.env.ezelkg	Link edit control statements generated from link edit parts	DSORG=PO, RECFM=FB, BLKSIZE=3120, LRECL=80	All z/OS
SYSLMOD	cghlq.env.LOAD	Load library	DSORG=PO, RECFM=U, BLKSIZE=6144, LRECL=0	All z/OS

List of Program Preparation Steps after Program Generation

Rational COBOL Runtime supports program preparation and installation in the z/OS environments using build scripts shipped with Rational COBOL Runtime. You must perform the steps listed in Table 14 before you can run your program in an z/OS target environment.

Table 14. Preparation Steps for z/OS Environments

Preparation Step	Environment
Transfer from workstation to the host	All
DB2 precompile	DB2 use only
CICS translation	CICS only
COBOL compile	All
Link	All
Bind	DB2 use only. A bind is also required if the first program in the run unit specifies a DB2 work database for IMS/VS

Additionally, for CICS and IMS environments, you must define your program and transactions to the environment:

- For CICS, you do this using the Resource Definition Online (RDO) PROGRAM and TRANSACTION entries. For information on CICS entries, see Chapter 11, “Preparing and Running a Generated Program in CICS.”
- For IMS, define your program and transactions through the IMS system definition. For information on the IMS system definition, see Chapter 14, “Preparing and Running Generated Programs in IMS/VS and IMS BMP,” on page 107.

Deploying generated code to USS

The setup for deploying generated Java code in USS is the same as for Windows. Please see "Setting up the J2EE runtime environment for EGL-generated code" in the *EGL Generation Guide*.

Output of Generation

After you generate a program, there are a number of objects that must be transferred to the z/OS host system and then prepared before you can run the program. During generation, EGL creates a build plan that controls the preparation process through the use of build scripts. By default, the build scripts do the following:

- Do not save the generated program source code or MFS source.
- Save the output of the preparation process (the DBRM, the object modules, and the load modules) as members in PDS data sets on the z/OS host. You control the high-level qualifier of the PDS data sets by setting the **projectID** build descriptor option.
- Save the object modules, link edit file, and the bind control file because these files are needed to recreate a load module without having to generate the program again.
- Save the CICS entries because they are needed to install the program in CICS.
- Save the sample runtime JCL for z/OS batch and IMS BMP programs.

You cannot save a load module in a workstation repository and then restore it to a z/OS host system. However, you can save the object deck, link edit file, and bind control file and then relink and bind the object deck in a production z/OS environment.

If you want to save the generated source code, you must modify the FDABCL, FDABPTCL, FDABTCL, FDAACL, FDAMFS, FDAPCL, FDAPTCL, and FDATCL build scripts. There are instructions in the build scripts on how to do this by removing the comment tag from certain lines and commenting others.

The following rules apply to using objects generated for one environment in a different environment:

- Main programs cannot be generated for one environment and used in a different environment.
- In general, FormGroup objects cannot be generated for one environment and used in a different environment. However, if you generate a FormGroup for IMS BMP or z/OS batch and specify the **formServicePgmType="ALL"** build descriptor option, you can use the FormGroup output for the IMS/VS, IMS BMP, and z/OS batch environments because this causes generation of all the output required to support MFS, GSAM, and SEQ print files. However, you must ensure that the resource association information is identical for IMS/VS and IMS BMP when using the MFS print forms and is identical for IMS BMP and z/OS batch when using GSAM or SEQ print forms.
- DataTables generated and prepared in one environment (whether CICS, z/OS batch, or IMS) can be used in another environment on the same system.
- A CICS application can call a common batch application as long as it does not perform any file I/O. You must call the application using the DYNAMIC and OSLINK linkage options and the called program must be generated for the z/OS batch environment.

- An IMS or IMS BMP application cannot call a common batch application.
- CICS and IMS applications can use common libraries under the following conditions:
 - The library must be generated for the z/OS Batch environment.
 - The library cannot perform any SQL or file I/O.

Table 15 provides information about the types of files produced by generation, including:

- Type of object produced
- Low-level qualifiers of the default PDS name to which the object is written if the build scripts are customized to save the generated files
- How the member name is derived
- Runtime environments for which the object is produced
- Whether production is controlled by a COBOL build descriptor option
- Whether the object can be modified after generation is performed

A description of each object begins on page 77.

For additional information on generation output, refer to the *EGL Generation Guide* in the help system.

You can specify an **alias** for a program, DataTable, or FormGroup, and that alias is used for generated output. If you do not specify an alias, the default value is the name of the part truncated to the requirements of the target environment.

The name given to the output includes the alias or the default name, as shown by *alias* in the next table.

A bind control file is always generated and used in preparation for programs that access an SQL database. You can specify your own bind control part to be used to generate the bind control file using the **bind** build descriptor option, or you can develop a bind control part with the same name as the program part. Otherwise, a default bind control part is generated.

Table 15. Objects Generated for Programs, Libraries, or Services and Transferred to the z/OS Host by the Build Scripts

File Type	PDS Low-level Qualifier	PDS Member Name	File Name on Workstation	z/OS Runtime Environment	Build Descriptor Option	Modifiable
COBOL program	EZESRC	<i>alias</i>	<i>alias.cbl</i>	All	None	No
Sample runtime JCL	EZEJCLX	<i>alias</i>	<i>alias.jcx</i>	z/OS batch IMS BMP	genRunFile	Yes
Bind command	EZEBIND	<i>alias</i>	<i>alias.bnd</i>	All	bind	Yes
Link Edit File generated automatically by EGL	EZELINK	<i>alias</i>	<i>alias.led</i>	All	None	Yes

Table 15. Objects Generated for Programs, Libraries, or Services and Transferred to the z/OS Host by the Build Scripts (continued)

File Type	PDS Low-level Qualifier	PDS Member Name	File Name on Workstation	z/OS Runtime Environment	Build Descriptor Option	Modifiable
Link edit control statements generated from link edit parts	EZELKG	<i>alias</i>	<i>alias.lkg</i>	All	linkedit	Yes
Build Plan	Not applicable (see note 1)	Not applicable	<i>aliasBuildPlan.xml</i>	All	prep	No
CICS Entry (See note 4)	EZEPPT	<i>alias</i>	<i>alias.ppt</i>	CICS	cicsEntries	Review and possible modification required
CICS Entry (See note 2)	EZEPCT	<i>alias</i>	<i>alias.pct</i>	CICS	cicsEntries startTransactionID restartTransactionID	Review and possible modification required

Table 16. Objects Generated for DataTables and Transferred to a z/OS Host by the Build Scripts

File Type	PDS Low-level Qualifier	PDS Member Name	z/OS Runtime Environment	Build Descriptor Option	Modifiable
DataTable COBOL program	EZESRC	<i>alias.cbl</i>	All	genDataTables	No

Table 17. Objects Generated for FormGroups and Transferred to a z/OS Host by the Build Scripts

File Type	PDS Low-level Qualifier	PDS Member Name	File Name on Workstation	z/OS Runtime Environment	Build Descriptor Option	Modifiable
Online print services program (see note 3)	EZESRC	<i>alias</i>	<i>alias.cbl</i>	CICS	genFormGroup, genHelpFormGroup	No
Batch print services program (see note 3)	EZESRC	<i>aliasP1</i>	<i>aliasP1.cbl</i>	z/OS batch, IMS BMP	genFormGroup, genHelpFormGroup, formServicePgmType	No
Form group format module (see note 5)	EZEFOBJ	<i>aliasFM</i>	<i>aliasFM.fmt</i>	z/OS CICS, IMS/VS	genFormGroup , genHelpFormGroupformServicePgmType	No
MFS print services COBOL program (see note 3)	EZESRC	<i>alias</i>	<i>alias.cbl</i>	IMS/VS IMS BMP	genFormGroup, formServicePgmType	No
MFS control blocks	EZEMFS	<i>alias</i>	<i>alias.mfs</i>	IMS/VS IMS BMP	formServicePgmType, genFormGroup, genHelpFormGroup	No

Table 17. Objects Generated for FormGroups and Transferred to a z/OS Host by the Build Scripts (continued)

File Type	PDS Low-level Qualifier	PDS Member Name	File Name on Workstation	z/OS Runtime Environment	Build Descriptor Option	Modifiable
COBOL copybook for MFS MID/MOD layout	EZECOPY	<i>alias</i>	<i>alias.cpy</i>	IMS/VS IMS BMP	formServicePgmType, genFormGroup, genHelpFormGroup	No

Notes:

1. Build plans are not transferred to the host. They define what needs to be sent to the host. Specifically, the build plan includes the name of a build script that runs on the build server. The build script also contains substitution variable values that are used for substitution in the build script.
For additional details, refer to the *EGL Generation Guide*.
2. If you specify the **cicsEntries="RDO"** build descriptor option, the PROGRAM entries are placed in *alias.ppt*. The TRANSACTION entries are placed in *alais.pct*.
3. This object is produced only if the FormGroup contains print forms.
4. This object is produced for programs, FormGroups, and DataTables.
5. This object is produced only if the FormGroup contains text forms.

Objects Generated for Programs

Application COBOL Program

The generated program is a COBOL program that contains the following:

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

The program control logic performs the following functions for a program, as needed:

- Initialization
- Cleanup at end of program
- Error reporting
- Segmentation support, including saving data before and restoring after a **converse** statement
- Transfer of control

Sample Runtime JCL

The generator produces sample runtime JCL for running programs in the z/OS batch and IMS BMP environments when the **genRunFile** build descriptor option is set to YES during program generation. Each person using the JCL must provide a JOB statement.

The JCL is produced from model JCL templates that you can modify to enforce customer data set naming conventions. For more information about modifying the sample templates, refer to the *EGL Generation Guide*.

The JCL might not be complete and should be reviewed and modified if necessary before being used. For example, the JCL for the generated program does not contain any DD statements for data sets used by other programs or that can be started by a **call** or **transfer** statement. Comments in the JCL indicate where DD

statements for these programs need to be added. To build the final JCL needed to run a set of programs as a run unit, you should edit the program JCL and include the DD statements for invoked programs with the JCL for the first main program. You might need to add DD statements for files that are specified during run time with the **resourceAssociation** record-specific variable or with the **converseVar.printerAssociation** system variable.

Bind Commands

Bind commands are required for an SQL program. The bind commands either reside in a bind control part that has the same name as the program or, you can specify the bind control part using the **bind** build descriptor option.

You are not required to supply a bind control part. If one is not supplied, EGL generates a default bind control part that may or may not meet the requirements of the program.

The bind control part generated by default cannot be affected by users. However, bind control parts provided by the user may contain references to symbolic parameters which get substituted at generation time.

Link Edit File

Link edit files are required for each program, DataTable program, print services program, and FormGroup format module. For programs, EGL always generates a default link edit file called *alias.led*. If you need the program to be link edited differently (for example, to be statically linked with other programs), you can create a program-specific link edit part that has the same name as the program, or you can specify the name of the link edit part using the **linkedit** build descriptor option. If you create a program-specific link edit part, EGL generates this part as a file called *alias.lkg*.

The link edit part generated by default cannot be affected by users. However, link edit parts provided by the user may contain references to symbolic parameters which get substituted at generation time.

CICS Entries

If you set the **cicsEntries** build descriptor option to YES, the PPT or RDO DEFINE PROGRAM entries are generated for you for the following:

- Each program, library, or service
- Each DataTable program
- The print services program and FormGroup format module for each FormGroup

If you set the **cicsEntries** build descriptor option to YES, the PCT or RDO DEFINE TRANSACTION commands are generated for you for main programs using the transaction names from both the **startTransactionID** and **restartTransactionID** build descriptor options.

Objects Generated for DataTables

DataTable COBOL Program

The DataTable program is a COBOL program that contains the DataTable contents defined in program working storage. This object is produced when you set the **genDataTables** build descriptor option to YES. This allows DataTables to be generated independently of programs when the contents of a DataTable need to be changed.

Objects Generated for FormGroups

Online Print Services Program

The online print services program is a COBOL program that performs print I/O, output formatting, and SET operations for a generated online CICS program that prints output. This object is produced when you set the **genFormGroup** or **genHelpFormGroup** build descriptor options to YES during program generation.

Batch Print Services Program

The batch print services program is a COBOL program that formats data for line printers and writes the data to either the printer output file (directly to the printer or a QSAM file) or to a generalized sequential access method (GSAM) file. This program is used with programs that run in the z/OS batch or IMS BMP environments. This object is produced when you set the **genFormGroup** build descriptor option to YES and also specify (or default to) the **formServicePgmType="ALL"**, **formServicePgmType="SEQ"**, or **formServicePgmType="GSAM"** build descriptor option.

FormGroup Format Module

The FormGroup format module is a generated structure that describes the layout for text forms in the FormGroup. The generator builds the structure as a z/OS object module for the CICS, IMS/VS, and IMS BMP environments. This object is produced when you set the **genFormGroup** or **genHelpFormGroup** build descriptor option to YES.

MFS Print Services Program

The MFS print service program is a COBOL program that performs print I/O, output formatting, and SET operations for a generated IMS/VS or IMS BMP program that prints output using MFS control blocks. This object is produced when you generate for the IMS/VS environment and set the **genFormGroup** or **genHelpFormGroup** build descriptor option to YES. It is also produced when you generate for the IMS BMP environment, set the **genFormGroup** build descriptor option to YES, and also specify (or default to) the **formServicePgmType="ALL"** or **formServicePgmType="MFS"** build descriptor option.

MFS Source

In the IMS environment, an MFS source file is generated at the same time as the FormGroup format module. The build server automatically compiles this MFS source to generate IMS format, input, and output messages for each device type defined.

COBOL Copybook for MFS MID/MOD Layout

The COBOL copybook provides the equivalent COBOL definition of the MFS MID and MOD layouts for text forms. You can use the COBOL copybook if you need to transfer to a non-EGL program using a **show** statement or transfer from a non-EGL program to an EGL program that specifies the **inputForm** property. If the FormGroup contains text forms, this object is produced when you generate for the IMS/VS environment and set the **genFormGroup** or **genHelpFormGroup** build descriptor option to YES. It is also produced when you generate for the IMS BMP environment, specify the **genFormGroup** build descriptor option, and also specify (or default to) the **formServicePgmType="ALL"** or **formServicePgmType="MFS"** build descriptor option .

Chapter 10. z/OS Builds

EGL generates the files needed to create an executable program. After creating these files, the generation process communicates with the build server on z/OS to transfer the files to the host and then initiate the appropriate builds (compiles, link-edits, binds, and so on) for these programs.

To control the build process, the EGL generation process creates an XML file called a build plan for each generated program. This build plan contains specific information that the build server uses when building the generated program.

The type of information that the build plan contains includes:

- The name of the build script that the build server invokes to process the build
- The location on the client workstation where the server places listings and diagnostics from the build tools (for example, the compiler or linkage editor)
- The generated program
- A list of dependent files for the build process (for example, the name of the link edit file or the bind file) containing information used by the build process
- A list of environment variables that are used to override the default VARS values specified in the Pseudo-JCL build script

The environment variables defined in the build plan are set using build descriptor options and symbolic parameters specified by the user during program generation.

Using the information in the build plan, the server invokes the build script overriding any predefined defaults in the pseudo-JCL build script with the appropriate values specified in the build plan.

Following the steps outlined in the build script, the build server transforms one set of files into another by invoking tools such as compilers and link editors. For example, using a build script, the build server might transform a COBOL source file into an object file. Another build script might perform the database bind.

After the build is finished, the build server places the listings and diagnostics from the build process in the location specified in the build plan or build script.

Prepared output is placed into PDSs on the build server machine. The high level and middle qualifiers of the PDS are controlled by the **projectID** and **system** build descriptor options. The low level qualifiers are controlled by the type of output.

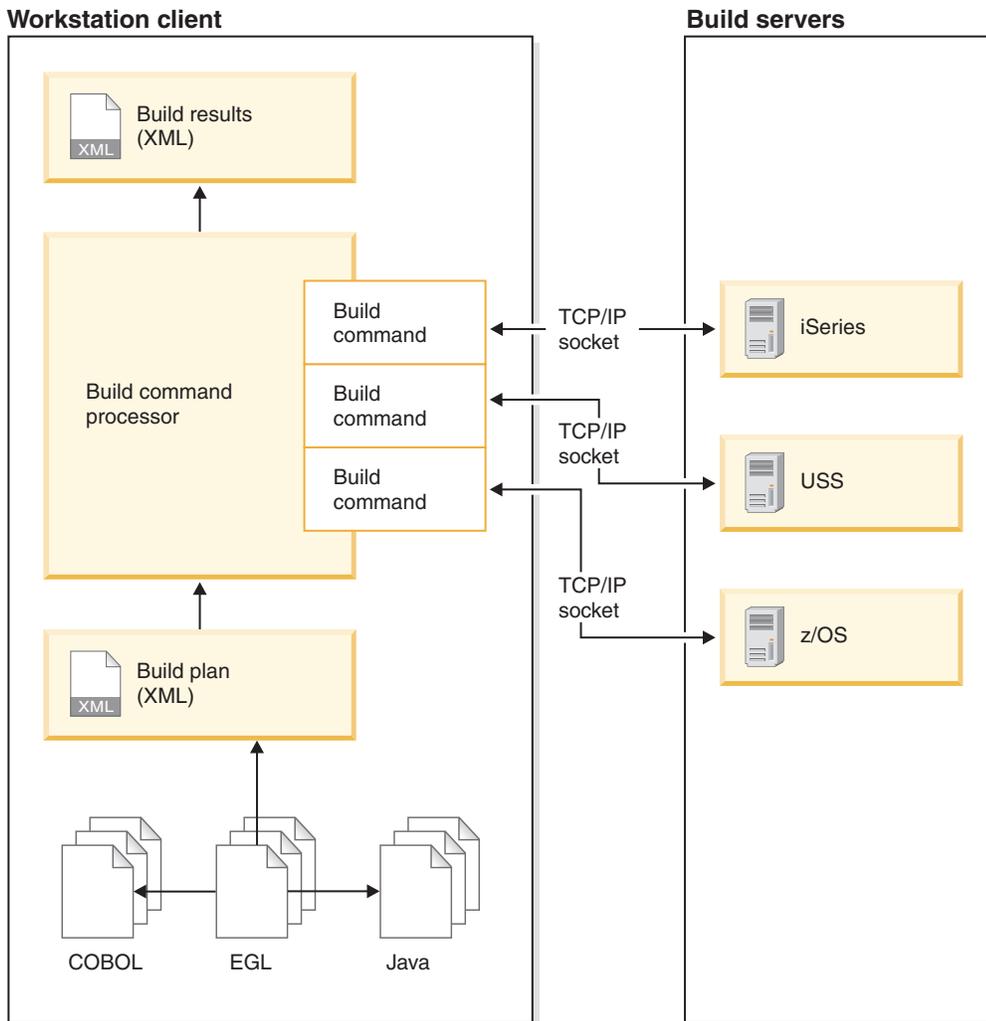


Figure 10. z/OS Build Process

z/OS Build Server

On z/OS, you can configure the build server to perform z/OS or USS builds. If you need both builds, then you need to start two build servers, each listening on a unique TCP/IP port for each type.

The Remote Build server performs the following tasks:

- Receives build transactions and files.
- Performs character conversions.
- Runs builds within its environment.
- Optionally collects and returns results to the client.

In z/OS, the server load module CCUBLDS receives client build transactions. CCUBLDS triggers the JCL member CCUMVS, which executes the CCUBLDW module. CCUBLDW processes your build scripts.

For USS operations, the server load modules CCUMAIN and CCUBLDS run in z/OS. CCUBLDS triggers the JCL member CCUUSS, which starts the USS shell script ccubldw. The ccubldw script starts the executable ccubldw, which processes build transactions.

Starting a z/OS Build Server

The z/OS build server, CCUBLDS, is an z/OS load module that you can run as a batch program.

```
//CCUBLDS JOB (ACCT#),'TEST',REGION=0M,
// CLASS=0,MSGCLASS=T
//-----
/* PROGRAM: CCUMAIN
/* JCL to start CCU z/OS Build Server
/*
/* COPYRIGHT: Copyright (C) International Business
/*          Corp. 2001
/*
/* DISCLAIMER OF WARRANTIES:
/* The following enclosed code is sample code created
/* by IBM Corporation. This sample code is not part
/* any standard product and is provided to you solely
/* for the purpose of assisting you in the development
/* of your applications. The code is provide "AS IS",
/* without warranty of any kind. IBM shall not be
/* liable for any damages arising out of your use
/* of the sample code
/*-----
/* Some dataset names may need to be modified
/* according to your system's customization
/*-----
//RUNPGM EXEC PGM=CCUMAIN,DYNAMNBR=30,REGION=7400K,TIME=NOLIMIT,
// PARM='-p 4112 -a 2 -n 3 -q 20 -T 20'
//STEPLIB DD DSN=CUST.UCCBLD.LOAD,DISP=SHR
//CCUWJCL DD DISP=SHR,DSN=install-prefix.SELASAMP(CCUMVS)
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//CCUBLOG DD SYSOUT=*
//
```

Figure 11. An example of the JCL needed to start the build server for z/OS

The CCUBLDS job initiates a new job for each build transaction. The sample JCL for that new job is in member CCUMVS of the installation data set whose low-level qualifier is SELASAMP. The server is multi-threaded, so these jobs run concurrently and are independent of each other. The number of concurrent jobs running at any one time is limited by system resources (such as initiators).

The build server receives commands and files, performs character conversions, sets up the environment, runs builds within this environment, collects the results and returns the results.

See the program directory for Rational COBOL Runtime for zSeries for additional information on customizing the following sample JCL members in the dataset whose low level qualifier is SELASAMP:

CCUMVS

Submitted in every build.

CCURUNM

Submitted to run the build server (CCUMAIN program).

You start a build server by using z/OS JCL commands. The syntax for the parameters line is as follows:

Syntax: // PARM= '-p <portno> [-V ...] [-a {2|1|0}] [-n <n>] [-q <q>] [-t] [-T <n>]'

where:

- p Specifies the port number (portno) to which the server listens to communicate with the clients.
- V Specifies the verbosity level of the server. You may specify this parameter up to three times (maximum verbosity).

For example, to increase the verbosity to the maximum, you specify -V -V -V.

- a Specifies the authentication mode of the CCUBLDS server. The server state is either 'A' (APF authorized) or 'U' (not APF authorized).
 - 2 Server state: A. The user submitting the build transaction must specify a valid user ID and password when the user initiates a build by using the remote build client. The server performs the build transaction under the access and authority of this user ID. Mode 2 is the default.
 - 1 Server state: A. The user submitting the build transaction can provide a valid user ID and password. The server performs the build transaction under the access and authority of this user. If the user does not provide a user ID and password, the build transaction is performed under the access and authority of the user ID assigned to the build server job.
 - 0 Server state: A or U. If U, APF-authorized build programs will fail. If the user submitting the build transaction specifies a TSO user ID and password, the server ignores them and the build transaction is performed under the access and authority of the user ID assigned to the build server job.

If you start the server on z/OS from an APF-authorized library (this is required in modes 1 and 2 but is optional in mode 0), the server state is authorized ('A') and the build script can specify an APF authorized program as the executable.

Notes:

1. For additional information about installing code in an APF-authorized library to allow users to run builds under the authority of their userid, see the program directory for Rational COBOL Runtime.
 2. In this case, the build script can also specify non-APF authorized programs. However, in a multistep JCL script, an authorized program cannot be executed after an unauthorized program.
 3. If the server is not started from an APF-authorized library, the server state is not authorized ('U') and the build script can specify only non-APF authorized programs as executables.
- n Specifies the number of concurrent builds. The default is 1. Set n equal to the number of concurrent builds you want to allow. Once there are n number of concurrent builds running, the build server queues any additional requests and submits them on a first come first served basis as builds are completed.
 - q Specifies the size of the queue (q) for clients. The default is 10. Each queued client uses a TCP/IP socket. Therefore setting this too high may require more sockets than are available, causing unpredictable results. If the queue is full, subsequent clients are rejected by the server. However, the build client automatically retries the build in that case.

- t Starts tracing of this server job and writes output to STDOUT. This parameter is normally used only for debugging.
- T Specifies the number of minutes the build server will wait for a started child process (CCUBLDW) to complete. If the system is overloaded, increase this value. The default is 5.

Note: See the program directory for Rational COBOL Runtime for zSeries for information about modifying the JCL necessary to start the USS and z/OS build servers

Starting a USS Build Server

You start the USS build server the same way you start the z/OS build server, except with a different dataset allocated by DD name CCUWJCL. This difference is reflected in the CCURUN and CCURUNU JCL customized at installation. The sample JCL CCURUNU needs to be modified just as CCURUN.

The CCUWJCL DD name uses the JCL member CCUUSS. As found in the installation data set whose low-level qualifier is SELASAMP, that member acts as a template in submitting build transactions to USS using the BPXBATCH utility to submit the USS shell script ccubldw.sh.

The build server creates temporary datasets and directories in the directory where the program is initiated. It is important that the ID that starts the server has the appropriate authority to create these datasets and directories otherwise the server will not initiate properly and all transactions will fail.

Alternatively, you can avoid using the build server and use your already compiled Java code from the EGL workbench. You can move the code to USS or WAS by exporting the resulting .ear, .jar, or .war file and sending that file to USS or WAS.

Stopping servers

To stop an z/OS server, cancel the job that was used to start it.

Configuring a build server

To configure a build server, you must modify members of the installation data set whose low-level qualifier is SELASAMP. Those members contain JCL and are named as follows:

- CCUMVS (for z/OS builds)
- CCUUSS (for USS builds)

Note: See the program directory for Rational COBOL Runtime for zSeries for information about configuring the USS and z/OS build servers.

Working with Build Scripts

There is a fundamental difference between build scripts on z/OS and build scripts on USS. Build scripts on z/OS must be text files and must be written in Pseudo-JCL. On USS, you can use any executable file as a build script and the file can be either text or binary.

Working with z/OS Build Scripts

The build script processed by the z/OS server is always a text file written in Pseudo-JCL. It is specified in one of two ways. If the build script is not specified as

part of the build command, then the server looks for it as a member of the PDS specified by the ddname CCUPROC for the server job. This PDS must be of RECFM=FB, LRECL=80.

The build script is parsed by the server. From the parsed results, the server allocates the specified DD names and data sets; it then executes the programs dynamically.

On z/OS, the server also uses the JCL to determine where to store the files involved in an z/OS build.

EGL uses and Rational COBOL Runtime provides build scripts in the PDS specified by DD name CCUPROC in the CCUMVS JCL. These build scripts are the defaults specified in the EGL generated build plans. The member names are FDABCL, FDABIND, FDABPTCL, FDABTCL, FDAACL, FDALINK, FDAMFS, FDAPCL, FDAPTCL, and FDATCL.

These must be members in the PDS specified in the CCUPROC DD card in the JCL used to invoke a build transaction (see the previous section). The members provide the following functions:

FDABCL

Compile and link the generated z/OS batch, IMS/VS, and IMS BMP programs.

FDABIND

Bind generated programs that contain DB2 statements.

FDABPTCL

DB2 precompile, then translate for CICS, compile, and link the generated z/OS batch programs making CICS EXCI calls.

FDABTCL

Translate for CICS, compile, and link the generated z/OS batch programs making CICS EXCI calls.

FDAACL

Compile and link the generated COBOL source for print services programs or DataTable programs that do not contain CICS or DB2 commands.

FDALINK

Link the generated FormGroup format module.

FDAMFS

Invoke the MFS utilities to prepare MFS source for execution in IMS/VS or IMS BMP environments.

FDAPCL

DB2 precompile, compile, and link the generated z/OS batch, IMS/VS, or IMS BMP programs that contain DB2 statements.

FDAPTCL

DB2 precompile, CICS translation, compile, and link the generated CICS COBOL programs that contain DB2 statements.

FDATCL

CICS translation, compile, and link the generated CICS COBOL programs that do not contain DB2 statements.

To override the default build scripts, use the symbolic parameter `DISTBUILD_BUILD_SCRIPT`. To identify the PDS from which to access build scripts at build time, specify the PDS name in the symbolic parameter `BUILD_SCRIPT_LIBRARY`.

Refer to the *EGL Generation Guide* in the EGL help system for more information on how to use symbolic parameters during generation.

Writing a JCL build script

JCL build scripts must be written using Pseudo-JCL. The best starting point for a JCL build script is an existing JCL fragment that is used for transforming inputs into output. For example, suppose you want to create a build script that compiles a COBOL source file into an OBJECT file using a z/OS compiler. You probably already have JCL that can be submitted as a batch job that does this.

When you create a build script for the z/OS environment, you specify Pseudo-JCL statements. See "Modifying EGL build scripts for z/OS" and "Pseudo-JCL syntax" in the *EGL Generation Guide*.

For more information about JCL syntax, refer to the *JCL User's Guide* and *JCL Reference* for your version of z/OS.

File Name Conversions for z/OS

Workstation file names are converted to z/OS host PDS names and member names by the z/OS build server according to the following rules:

- The directory path of a file name is not used. The end of a directory path of a file name is specified by a slash or left parenthesis ("/", "(", or "\"). All characters of a file name up to and including the rightmost slash or left parenthesis are discarded.
- Lowercase characters are converted to uppercase characters.
- The file extension is stripped, including the separating period. The extension, minus the period, is used by the z/OS server to direct the file to particular data sets according to user-specified syntax in the JCL build scripts.
- The remaining name is truncated to a maximum of 8 characters.
- Names must contain characters that are valid in z/OS. z/OS allows the following characters:
0123456789ABCDEFGHIJKLMNPOQRSTUVWXYZ\$#@#
The name must begin with an alphabetic character.
- Underscore characters (_) in a file name are converted to at signs (@).

The following are examples of how a workstation name is converted:

- A file name of `src\build\fhbldobj.CBL` is converted to `FHBLDOBJ` on z/OS.
- A file name of `src/build/fhbtruncate.cbl` is converted to `FHBTRUNC` on z/OS.

In both of these examples, the `.CBL` or `.cbl` is removed. The z/OS server uses the resulting extension to resolve and possibly allocate the z/OS data sets needed for the build process. The extensions are required for files that participate in an z/OS build.

Converting JCL to Pseudo-JCL

The following is a JCL procedure for a z/OS compile and link:

```

//*****
//* JCL Procedure - COBOL COMPILE AND LINK-EDIT
//*****
//*
//ELACL PROC CGHLQ='USER',
//      COBCOMP='SYS1.IGY.SIGYCOMP',
//      COBLIB='SYS1.SCEELKED',
//      ELA='ELA.V6R0M1',
//      DATA='31',
//      ENV='ZOSCICS',
//      MBR=PGMA,
//      RESLIB='SYS1.RESLIB',
//      RGN=1024K,
//      SOUT='*',
//      WSPC=500 ,
//
//* PARAMETERS:
//*   CGHLQ  = COBOL GENERATION USER DATA SET HIGH LEVEL QUALIFIER
//*   COBCOMP = COBOL COMPILER LIBRARY
//*   COBLIB  = LE RUN TIME LIBRARY
//*   ELA     = EGL SERVER HIGH LEVEL QUALIFIER
//*   DATA   = COMPILE OPTION FOR PLACING WORKING STORAGE
//*           ABOVE 16M LINE
//*   ENV     = COBOL GENERATION USER DATA SET ENVIRONMENT QUALIFIER
//*           (SHOULD BE EQUAL TO GENERATION TARGET ENVIRONMENT)
//*   MBR     = SOURCE NAME
//*   RESLIB  = IMS RESLIB LIBRARY
//*   RGN     = REGION SIZE
//*   SOUT    = SYSOUT ASSIGNMENT
//*   WSPC    = PRIMARY AND SECONDARY SPACE ALLOCATION
//*
//*****
//*           COMPILE THE COBOL PROGRAM
//*****
//*
//C           EXEC PGM=IGYCRCTL,REGION=&RGN,
//           PARM=(NOSEQ,QUOTE,OFFSET,LIB,RENT,NODYNAM,DBCS,OPT,
//           'TRUNC(BIN)', 'NUMPROC(NOPFD)',NOCMPR2, 'DATA(&DATA)')
//STEPLIB DD DISP=SHR,DSN=&COBCOMP
//SYSIN   DD DISP=SHR,DSN=&CGHLQ..&ENV..EZESRC(&MBR)
//SYSLIB  DD DISP=SHR,DSN=&ELA..SELACOPY
//SYSLIN  DD DISP=(MOD,PASS),DSN=&&LOADSET,UNIT=VIO,
//           SPACE=(800,(&WSPC,&WSPC))
//SYSPRINT DD SYSOUT=&SOUT,DCB=BLKSIZE=13300
//SYSUDUMP DD SYSOUT=&SOUT,DCB=BLKSIZE=13300
//SYSUT1  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//SYSUT2  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//SYSUT3  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//SYSUT4  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//SYSUT5  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//SYSUT6  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//SYSUT7  DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
//*
//*****
//*           LINK-EDIT THE COBOL PROGRAM
//*           IF THE RETURN CODE ON ALL PREVIOUS STEPS IS 4 OR LESS
//*****
//*
//L           EXEC PGM=IEWL,COND=(5,LT,C),REGION=&RGN,
//           PARM='RENT,REUS,LIST,XREF,MAP,AMODE(31),RMODE(ANY)'
//SYSLIB  DD DISP=SHR,DSN=&COBLIB
//           DD DISP=SHR,DSN=&RESLIB
//SELALMD DD DISP=SHR,DSN=&ELA..SELALMD
//SYSLIN  DD DISP=(OLD,DELETE),DSN=&&LOADSET
//           DD DDNAME=SYSIN
//SYSLMD  DD DISP=SHR,DSN=&CGHLQ..&ENV..LOAD(&MBR)

```

```
//SYSPRINT DD SYSOUT=&SOUT,DCB=BLKSIZE=13300
//SYSUDUMP DD SYSOUT=&SOUT,DCB=BLKSIZE=13300
//SYSUT1 DD SPACE=(1024,(&WSPC,&WSPC)),UNIT=VIO
```

The first step in converting the JCL fragment is to recognize the intent for each of the data sets and DD names. For this COBOL compiler example, the SYSIN DD name needs to be associated with the source file, the SYSLIN DD name needs to be associated with the object file, and so on.

In each of these cases, the build script must tell the server where to pick up the input files before the execution of the specified program (PGM=IGYCRCTL) and where to put the output files after the execution of the specified program.

Assume that your source files have the extension .cbl. You allocate a data set to the SYSIN DD name to contain a source file with a .cbl extension. You specify the DCB, UNIT, DISP, and SPACE attributes to dynamically create this data set every time this build script is invoked. You add CCUEXT=CBL to indicate that the file content comes from an input file with an extension of .cbl.

For the SYSPRINT DD statement, use the CCUEXT parameter to tell the z/OS build server what you want to have done with the COBOL compiler listing. In the example, CCUEXT=&CCUEXTC so that the value is set from the default Pseudo-JCL build script parameter CCUEXTC. The value CCUOUT indicates that you want the listing returned to the client as a file with a name based on the DD name.

The following JCL build script is the result of converting the JCL procedure.

```
//*****
//* BUILD SCRIPT - COBOL COMPILE AND LINK-EDIT
//*****
//*
//DEFAULTS VARS CGHLQ=USER,
//              COBCOMP=SYS1.IGY.V3R1M0.SIGYCOMP,
//              COBLIB=SYS1.SCEELKED,
//              COBLISTPARMS=OFFSET&COMMA.NOLIST&COMMA.MAP,
//              ELA=ELA.V6R0M1;,
//              DATA=31,
//              SYSTEM=ZOSCICS,
//              MBR=PGMA,
//              RGN=4096K
//              CCUEXTC=CCUOUT,
//              CCUEXTL=CCUOUT,
//              SOUT=*,
//              DBCS=&COMMA.DBCS
//              WSPC=2500
//*
//* PARAMETERS:
//* CGHLQ = COBOL GENERATION USER DATA SET HIGH LEVEL QUALIFIER
//* COBCOMP = COBOL COMPILER LIBRARY
//* COBLIB = LE RUN TIME LIBRARY
//* COBLISTPARMS = LISTING OPTIONS FOR COBOL COMPILER
//* ELA = RATIONAL COBOL RUNTIME HIGH LEVEL QUALIFIER
//* DATA = COMPILE OPTION FOR PLACING WORKING STORAGE
//*        ABOVE 16M LINE
//* DBCS = COMPILE OPTION FOR INDICATING SOURCE CONTAINS DBCS
//*        CHARACTERS
//* SYSTEM = SYSTEM GENERATING FOR. USED AS USER DATASET MIDDLE
//*        QUALIFIER
//* MBR = SOURCE NAME
//* RGN = REGION SIZE
//* CCUEXTC = CCUEXT VALUE FOR COMPILE PRINTOUTS RETURNED TO
//*        CLIENT.
```

```

/**          CCUOUT=RETURN TO CLIENT AS FILE NAMED BY DDNAME
/**          CCUSTD=RETURN TO CLIENT AS STANDARD OUT
/**          CCUERR=RETURN TO CLIENT AS STANDARD ERROR
/**          CCUEXTL = CCUEXT VALUE FOR LINK PRINTOUTS RETURNED TO CLIENT
/**          CCUOUT=RETURN TO CLIENT AS FILE NAMED BY DDNAME
/**          CCUSTD=RETURN TO CLIENT AS STANDARD OUT
/**          CCUERR=RETURN TO CLIENT AS STANDARD ERROR
/**          SOUT    = SYSOUT ASSIGNMENT IF A SYSOUT FILE NOT RETURNED
/**                TO CLIENT
/**          WSPC    = PRIMARY AND SECONDARY SPACE ALLOCATION
/**
/*****
/**          COMPILE THE COBOL PROGRAM
/*****
/**
/**C          EXEC PGM=IGYCRCTL,REGION=&RGN,
/**          PARM='NOSEQ,QUOTE,LIB,RENT,NODYNAM,OPT&DBCS,
/**          TRUNC(BIN),NUMPROC(NOPFD),&COBLISTPARMS.,DATA(&DATA) '
/**STEPLIB DD DISP=SHR,DSN=&COBCOMP
/** COBOL SOURCE CODE UPLOADED FROM CLIENT (&MBR.CBL)
/**SYSIN DD CCUEXT=CBL,DISP=(NEW,DELETE),
/**          UNIT=SYSDA,SPACE=(TRK,(10,10)),
/**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
/**SYSLIB DD DISP=SHR,DSN=&ELA..SELACOPY
/**SYSLIN DD DISP=SHR,DSN=&CGHLQ..&SYSTEM..OBJECT(&MBR),ENQ=YES
/** RETURN COMPILER LISTING TO CLIENT AS FILE &PREFIX.C.SYSPRINT
/**SYSPRINT DD CCUEXT=&CCUEXTC,DISP=(NEW,DELETE),
/**          UNIT=VIO,SPACE=(CYL,(5,5)),
/**          DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
/**SYSUT1 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**SYSUT2 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**SYSUT3 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**SYSUT4 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**SYSUT5 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**SYSUT6 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**SYSUT7 DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=VIO
/**
/*****
/**          LINK-EDIT THE COBOL PROGRAM
/**          IF THE RETURN CODE ON ALL PREVIOUS STEPS IS 4 OR LESS
/*****
/**
/**L          EXEC PGM=IEWL,COND=(5,LT,C),REGION=&RGN,
/**          PARM='RENT,REUS,LIST,XREF,MAP,AMODE(&DATA),RMODE(ANY) '
/**SYSLIB DD DISP=SHR,DSN=&COBLIB
/**SELALMD DD DISP=SHR,DSN=&ELA..SELALMD
/**OBJLIB DD DISP=SHR,DSN=&CGHLQ..&SYSTEM..OBJECT
/** LINK EDIT CONTROL FILE UPLOADED FROM CLIENT (&MBR.LED)
/**SYSLIN DD CCUEXT=LED,DISP=(NEW,DELETE),
/**          UNIT=SYSDA,SPACE=(TRK,(10,10)),
/**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
/**SYSMOD DD DISP=SHR,DSN=&CGHLQ..&SYSTEM..LOAD(&MBR),ENQ=YES
/** RETURN LINK EDIT LISTING TO CLIENT AS FILE &PREFIX.L.SYSPRINT
/**SYSPRINT DD CCUEXT=&CCUEXTL,DISP=(NEW,DELETE),
/**          UNIT=VIO,SPACE=(TRK,(30,10)),
/**          DCB=(RECFM=FB,LRECL=121,BLKSIZE=1210)
/**SYSUT1 DD SPACE=(1024,(&WSPC,&WSPC)),UNIT=VIO
/**

```

Chapter 11. Preparing and Running a Generated Program in CICS

This chapter describes the unique steps required to prepare and run a generated COBOL program in an CICS environment:

- Modifying CICS resource definitions
- Modifying CICS startup JCL
- Making new modules available
- Making programs resident
- Running programs

Modifying CICS Resource Definitions

The CICS environment uses resource definitions to identify startup parameters, transactions, programs, files, databases, transient data destinations, and system locations for proper operation. You must add to or modify these resource definitions to correctly identify all objects to be used in the new or changed program. When using CICS tables, the tables are compiled as assembler programs and stored in a runtime library. Some tables can also be maintained through an online facility as described in the resource definition online manual for your version of CICS. CICS requires that the online facility be used for PROGRAM and TRANSACTION entries.

Refer to the CICS resource definitions guide for additional information on providing definitions.

You can either write your own RDO PROGRAM and TRANSACTION entries or use the ones generated by EGL. However, you must create other resource definitions needed by your program, such as those required for transient data queues, files, or DB2.

Program Entries

The EGL COBOL generation process creates programs that must be defined, as a resource definition online (RDO) PROGRAM entry or by using dynamic program entries.

An entry is required for each EGL generated program. You can request that sample PPT or RDO entries be generated for you by specifying the **cicsEntries** build descriptor option at generation. However, the PPT entries are no longer supported by CICS.

Either the batch program DFHCSDUP utility or the resource definition online (RDO) CEDA DEFINE PROGRAM command can be used to define the programs to CICS.

If you specify **cicsEntries="RDO"**, CICS RDO DEFINE PROGRAM commands are generated for you for each program that requires an RDO PROGRAM entry. The build plan created during generation uploads the RDO command files to the z/OS library specified at generation.

The following example shows how to define the PROGRAM entries using the RDO CEDA transaction DEFINE PROGRAM command.

```
CEDA DEF PROG(progname) L(LE370) REL(NO) RES(NO) S(ENABLED) GROUP(xxxx)
```

The values shown for REL, RES, and S keywords are the default values and can be omitted from the command. RES(YES) might provide better performance for frequently used programs.

Transaction Entries

A CICS TRANSACTION entry contains the control information used by CICS for identifying and initializing a transaction. This entry is required by CICS to verify incoming requests to start transactions, and to supply information about the transaction such as the transaction priority, the security key, and the length of the transaction work area (TWA).

A CICS RDO TRANSACTION entry is required for each transaction code used to start an EGL generated program. If you specify **cicsEntries="RDO"**, CICS RDO DEFINE TRANSACTION commands are generated for you for main programs using the transaction names from both the **startTransactionID** and the **restartTransactionID** build descriptor options. The following example shows how to define the TRANSACTION entries using the RDO CEDA transaction DEFINE TRANSACTION:

```
CEDA DEF TR(tran) PROG(progname) ACTION(BACKOUT) DU(NO) RES(NO) TW(1024)
```

EGL generated programs can be started by a remote procedure call from some remote systems. The CICS supported mirror program DFHMIRS, normally invoked by the CPMI transaction is used during this remote procedure call. It:

1. Determines which server program should be given control
2. Builds the COMMAREA
3. Links to the defined server program via CICS LINK

CPMI is the CICS supplied default transaction code to invoke the CICS mirror program DFHMIRS. When using CPMI to start EGL programs, you must change the transaction definition for CPMI to specify a TWASIZE of at least 1024 bytes.

To avoid making changes to the CPMI definition in the CICS supplied group, it is recommended that you copy the CICS supplied CPMI definitions to a new group or create a unique transaction ID with the same characteristics as CPMI. The new transaction or copy of CPMI should be changed and verified to ensure the following values are set.

1. The twasize is 1024
2. The profile is DFHCICSA (CICS default would be DFHCICST (T for terminal))
3. The program invoked is DFHMIRS

Example:

```
DEFINE TRANSACTION(MYMI) PROGRAM(DFHMIRS) TWASIZE(1024) PROFILE(DFHCICSA)
```

Destination Control Table Entries

A CICS TDQUEUE entry is required for each program file that is assigned to a transient data queue. A TDQUEUE entry is also required for destinations specified as error destination queue names using the Rational COBOL Runtime diagnostic controller utility. The parameters for TDQUEUE entries depend on your destination type. There are intrapartition, extrapartition, indirect, and remote destinations. See "Using and Allocating Data Files in CICS" on page 41 for information about defining and managing program data files and "Defining

Transient Data Queues” on page 43 for information about defining the DCT entry for the error destination queue. Refer to appropriate CICS manuals for more information on TDQUEUE entries.

File Control Table Entries

A CICS FILE entry is required for each program file that is specified as file type VSAM. You must identify all FILE entries that might be referenced at run time. See “Using and Allocating Data Files in CICS” on page 41 for more information on defining and managing program data files in the CICS environment.

DB2 Entries

If the program running under a transaction accesses a DB2 database, then you must define a CICS DB2CONN entry to define the DB2 connection. You must also define a CICS DB2ENTRY entry to define the relationship of the transaction to the DB2 plan. If there is more than one transaction that uses the DB2 plan, you must define CICS DB2TRAN entries to define the relationship of the additional transactions to the DB2ENTRY. The information that you specify is the same as you specify for any CICS transaction, regardless of whether it is written in EGL or another language.

The following example shows the types information that you must specify:

```
CEDA DEF DB2CONN(connectionName) DB2ID(db2Subsystem)
CEDA DEF DB2ENTRY(entryDefName) TRANSID(tran) PLAN(planName)
CEDA DEF DB2TRAN(tranDefName) TRANSID(tran) ENTRY(entryDefName)
```

For more information on how to specify the parameters shown in the example, alternative ways of providing the equivalent information, and other parameters you can specify when you define these DB2 entries, see the CICS resource definition guide for your release of CICS.

Using Remote Programs, Transactions, or Files

Refer to the appropriate CICS manuals for information about defining remote programs, transactions, or files.

CICS Setup for Calling CICS Programs from z/OS Batch

You must set up the CICS region to receive EXCI calls. In particular, you must meet the following CICS requirements:

- Include IRCSTRT=YES in the CICS region.
- Install the CICS default group DFH\$EXCI or equivalent, as in the following example:

```
CEDA IN GR(DFH$EXCI) 5
```

CICS Setup for Calling z/OS Batch Programs in CICS

This section describes the setup that is necessary in the following case: an EGL program that runs under CICS calls an EGL program that was generated for z/OS batch but also runs under CICS.

To ensure that the call works, do as follows:

1. Include the EGL runtime dataset in the STEPLIB of the CICS region. The last qualifier of that dataset is SELALMD.

2. If the STGPROT setting in the region is STGPROT=YES, ensure that the following actions are done:
 - The transaction is defined to run in TASKDATAKey=CICS.
 - The program associated with the transaction is defined to run with EXECKEY=CICS.
 - Any subsequent called program that is invoked with a CICS LINK is defined to run with EXECKEY=CICS.
3. If the called, generated program accesses DB2, two copies of the load module are needed, each linked with a different DB2 interface module. Do as follows:
 - a. Link a copy of the program with the DB2 batch interface module (DSNALI) and place the load module in the STEPLIB of the batch job.
 - b. Link a second copy of the program with the CICS DB2 interface module (DSNCLI) and place the load module in the DFHRPL of the CICS region.If necessary, you can set up the EGL ZOSBATCH build scripts to run a relink step into the CICS load library.

Modifying CICS Startup JCL

You must include the load library where your generated programs reside in the DFHRPL DD concatenation. Your system administrator included the LE runtime libraries and the Rational COBOL Runtime load library in the DFHRPL DD concatenation when the Rational COBOL Runtime product was installed.

The CICS startup JCL might need to be modified to add or change allocations for files used by EGL-generated programs. These include VSAM files and extrapartition transient data destinations.

For VSAM data sets, it is not necessary to include allocations in the startup JCL if you specify the data set name and disposition in the CICS FILE entry for the file. CICS dynamically allocates the file at open time.

Making New Modules Available in the CICS Environment

After you generate a new version of a program, FormGroup, or DataTable you need to make the modules available to CICS.

For programs and FormGroups, you can use the CICS NEWCOPY command or the Rational COBOL Runtime new copy utility to cause the new copy of the program to be used the next time a load request is issued for the program. If you use the CICS NEWCOPY command for a FormGroup, you must issue the NEWCOPY for both the online print services program and the FormGroup format module.

For DataTables, you must use the Rational COBOL Runtime new copy utility to cause a fresh copy of the DataTable to be used the next time a load request is issued for the table. Do not use the CICS NEWCOPY command for DataTables. The Rational COBOL Runtime new copy utility sets a flag indicating that the new copy of the DataTable is to be used the next time a program loads the DataTable contents.

For more information on the Rational COBOL Runtime new copy utility, see “New Copy” on page 122.

Making Programs Resident

You can make frequently used programs or programs with high performance requirements resident to avoid the overhead of loading the programs when they are used. To aid in deciding which programs should be made resident, you can use CICS shutdown statistics to determine how often a generated program is loaded in a CICS region.

To make a program or FormGroup resident, specify the program as resident in the RDO entry for the program. To make a DataTable program resident, set the **resident** property to YES when you define the DataTable in EGL.

Running Programs under CICS

Either a main Text UI program or a main basic program generated for the z/OS CICS environment can be started with CICS facilities. Called programs can be started by another EGL program, by a non-EGL program, or through the remote CICS services.

Prior to running a generated program, the program user might be required to sign on to the CICS environment. Refer to CICS documentation for information about signing on.

Starting the Transaction in CICS

Any main program that is generated with a target environment of z/OS CICS can be started by entering the transaction code associated with the main program from a clear screen in CICS. Any main program that is started in any of the following ways must have a unique transaction code assigned to it:

- Directly in CICS
- By a **transfer to transaction** statement from another program
- By a **show** statement from another program
- By a **vgLib.startTransaction()** system function

The transaction code must be defined with an RDO TRANSACTION entry and be associated with the first program in the run unit.

Controlling Diagnostic Information in the CICS Environment

Rational COBOL Runtime provides a diagnostic controller utility for the CICS environment. This utility allows you to control the type of dump, the name of the error destination queue and journal number for error messages, and whether the transaction is disabled when a run unit error occurs. See “Diagnostic Control Options for z/OS CICS Systems” on page 125 for more information about the diagnostic controller utility.

Printing Diagnostic Messages in the CICS Environment

Rational COBOL Runtime provides a way to print diagnostic messages written to a transient data queue. See “Diagnostic Message Printing Utility” on page 124 for more information.

Chapter 12. Creating or Modifying Runtime JCL on z/OS Systems

This chapter contains the information you need to modify the sample runtime JCL created during program generation. You might need to modify the sample runtime JCL for the following reasons:

- EGL does not include DD statements in the JCL to allocate data sets accessed by programs called by or transferred to from the generated program.
- The generator does not include DD statements to allocate data sets accessed when the EGL program moves a value to the record-specific variable **resourceAssociation** or to the system variable **converseVar.printerAssociation**.
- The generator does not create any recovery or restart JCL.
- The sample JCL is based on the initial program in the run unit.

You need to ensure that the load libraries containing the initial program and any dynamically invoked programs are included in the STEPLIB concatenation unless you are using methods to put the load modules in memory. This includes program modules that are called dynamically or that receive control by a transfer and includes print services programs, FormGroup format modules, and DataTable programs.

Tailoring JCL before Generation

EGL creates sample runtime JCL for basic programs being generated for the z/OS batch or IMS BMP environments. The sample runtime JCL is based on templates that are installed in the following location:

sharedInstallationDirectory/plugins/com.ibm.etools.egl.generators.cobol_version/MVStemplates

sharedInstallationDirectory

The path to the directory where you installed Rational® COBOL Runtime for zSeries®.

version

The product version, for example, 6.0.0.

You can specify the location of site-specific templates by setting the **templateDir** build descriptor option.

Some of the reasons to tailor the JCL templates are as follows:

- Implementing your installation's data set naming conventions
- Adding DD statements to the STEPLIB concatenation
- Specifying a different DB2 subsystem

The sample JCL is shown in Chapter 13, "Preparing and Running Generated Programs in z/OS Batch," on page 101 and in Chapter 14, "Preparing and Running Generated Programs in IMS/VS and IMS BMP," on page 107.

The following table shows the relationship between the JCL templates used, the target environments, and the types of databases being used by the program.

Table 18. Runtime JCL Templates Based on Environment and Databases

JCL Template	Database	Calls CICS EXCI?	Environment
fda2mebe	None	No	z/OS batch
fda2mebx	None	Yes	z/OS batch
fda2mebd	DB2	No	z/OS batch
fda2mesx	DB2	Yes	z/OS batch
fda2mebb	DB2 and DL/I	n/a	z/OS batch
fda2mebc	DL/I	n/a	z/OS batch
fda2meia	DB2	n/a	IMS BMP
fda2meib	Without DB2	n/a	IMS BMP
fda2meba	Any, for called program	n/a	z/OS batch or IMS BMP

Table 19 shows the JCL templates that serve as models for DD statement generation for program-dependent files and databases.

Table 19. Model DD Statement for Program-Dependent Files and Databases

JCL Template	Contents
fda2msdi	QSAM input file
fda2msdo	QSAM output file
fda2mvsi	VSAM input file
fda2mvso	VSAM output file
fda2mgsi	GSAM input file
fda2mgso	GSAM output file
fda2mims	GSAM IMS dataset for IMS BMP
fda2mcal	Comment indicating where to insert DD statements for known transferred-to and called programs
fda2meza	Comment indicating where to insert DD statements for programs transferred-to using the system variable sysVar.transferName
fda2mezd	Comment indicating where to insert DD statements for data sets using the record-specific variable resourceAssociation or the system variable converseVar.printerAssociation
fda2mdli	Comment indicating where to insert DD statements for DL/I databases on z/OS batch

Modifying Runtime JCL

The sample runtime JCL for main basic programs contains EXEC statements to run a program or a cataloged procedure. The JCL for main basic programs does not include a JOB statement or the DD statements for data sets accessed by called or transferred-to programs. Before you use the JCL to run the program, you must do the following:

- Add a JOB statement.
- Insert missing DD statements as required. Comments in the generated JCL indicate where to insert the DD statements.

The sample runtime JCL for a called program contains only the DD statements that are required for the called program.

After generation, add the DD statements for any files required by called or transferred-to programs (including those named with **sysVar.transferName**) to the sample JCL for the main program. In addition, you must add DD statements for any files accessed by moving a value to the record-specific variable **resourceAssociation** or to the system variable **converseVar.printerAssociation**. You do not need to add DD statements for files that you access dynamically. You can also customize the sample runtime JCL with respect to specific data set name assignments, DCB information, output file space allocations, additional steps, and other relevant data.

The type of runtime JCL generated for a main basic program varies based on the types of databases used by the main program and whether the program uses CICS EXCI to call a program running in a CICS region, as shown in Table 18 on page 98. The generated runtime JCL does not consider the types of databases accessed by called or transferred-to programs, or whether the called or transferred-to program calls a program running in a CICS region. For example, if the main program does not use relational databases, but it calls or transfers to programs that use relational databases, you must modify the runtime JCL for the main program.

Consider the following situation:

- Program A is a main basic program that does not use relational databases.
- Program B is main basic program that accesses relational databases.
- Programs A and B are generated for the z/OS batch environment.
- Program A transfers to program B

Because program A does not use DB2, the JCL generated for program A is for a main basic program without DB2 access (as shown in Figure 12 on page 102). This JCL will not run correctly because program B requires DB2 to run. However, the JCL generated for program B is for an z/OS batch job with DB2 access (as shown in Figure 13 on page 103). The runtime JCL for program B can serve as a starting point for creating the JCL required to run program A. The following changes are required to the runtime JCL for program B:

- Change RUN PROG(APPLB) to RUN PROG(APPLA).
- Add any DD statements for files required by program A or other programs in the job step.

If program B is a called program and program A calls B rather than transferring to B, the runtime JCL for program B consists only of DD statements. In this situation, you need to create your own program JCL. Any one of the following can serve as a starting point for the JCL:

- The runtime JCL for another main program that accesses relational databases.
- The JCL template for the appropriate combination of DL/I and DB2.
- The examples shown in Chapter 13, “Preparing and Running Generated Programs in z/OS Batch,” on page 101 for the appropriate combination of DL/I and DB2.

You can avoid the modification just described if you include an I/O statement for an SQL table in the initial main program.

You must modify the JCL that is generated for the first main program in the job in the following additional situations:

- The first main program does not use DL/I and does not include a PSB, but calls or transfers to another program that uses DL/I.
- The first main program does not call a CICS program, but it calls or transfers to another program that calls a program in the CICS region.

If you get a JCL error for the runtime JCL, check the Generation Results view for the programs involved for any error messages related to JCL generation. In addition, ensure the tailoring that was done for the JCL templates is correct. Also check any changes you made when you customized the sample runtime JCL.

Chapter 13. Preparing and Running Generated Programs in z/OS Batch

This chapter describes the unique steps required to prepare a generated COBOL program to run in a z/OS batch environment:

- Running main programs
- Examples of runtime JCL
- Recovery and restart

For general information on preparing your program for the runtime environment, see Chapter 9, “Output of Program Generation on z/OS Systems.” For information on modifying the JCL, see Chapter 12, “Creating or Modifying Runtime JCL on z/OS Systems.”

Running Main Programs under z/OS Batch

A main basic program generated for the z/OS batch environment can be started by submitting JCL. Called programs can only be started by another EGL program or by a non-EGL program.

The EGL COBOL generation process creates sample runtime JCL for running programs in the z/OS batch environment. The generated JCL has same name as the program. If you set the **genRunFile** build descriptor option to "YES", sample JCL is created specifically for the program during program generation. The build plan uploads the sample runtime JCL to a z/OS partitioned data set (PDS).

The JCL might need to be modified to add data sets required by called or transferred-to programs. You also need to modify the JCL to add any data sets that are dynamically allocated with the *recordName.resourceAssociation* or **converseVar.printerAssociation** system variables. See Chapter 12, “Creating or Modifying Runtime JCL on z/OS Systems,” on page 97 for more information on modifying the sample runtime JCL.

If you get a JCL error for the runtime JCL, check the Generation Results view for the programs involved for any error messages related to JCL generation. In addition, ensure the tailoring that was done for the JCL templates is correct. Also check any changes you made when you customized the sample runtime JCL.

The following sections show JCL for different z/OS batch programs.

Examples of Runtime JCL for z/OS Batch Programs

The generated JCL in the following examples has these characteristics:

- The examples are based on the JCL templates shipped with EGL. Your actual JCL templates might differ if your system administrator has tailored them for your organization. Refer to the *EGL Generation Guide* in the EGL help system for more information about tailoring JCL templates.
- Lowercase text appears in the examples where a generic example name has been substituted for an actual program or data set name.
- EZEPRINT is always routed to SYSOUT=*

If you route EZEPRINT to a data set, you must use the following DCB attributes:

- LRECL=137, BLKSIZE=141, RECFM=VBA if the FormGroup does not contain any DBCS maps
- LRECL=654, BLKSIZE=658, RECFM=VBA if the FormGroup contains any DBCS maps

You cannot use FormGroups that do not have any DBCS forms with FormGroups that do have DBCS forms in the same job step.

Running a Main Basic Program with No Database Access

Figure 12 shows the JCL used to start a main basic program.

```
//jobname JOB ..... ,MSGCLASS=A
//stepnam EXEC PGM=appl-name,REGION=6M
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
// DD DSN=cgh1q.env.LOAD,DISP=SHR
//ELAPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//ELASNAP DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//EZEPRINT DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//SYSABOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//* Application specific DD statements
//file-name-1 DD .....
//file-name-n DD .....
```

Figure 12. JCL for Main Basic Program Run as z/OS Batch without DB2 or DL/I Access

If the program calls a CICS program, include the following DD statement in the STEPLIB:

```
// DD DSN=CICSTS.V3R1M0.CICS.SDFHEXCI,DISP=SHR
```

Running a Main Basic Program with DB2 Access

Figure 13 on page 103 shows the JCL used to start a main basic program that gains access to DB2 resources. The JCL must run the z/OS TSO terminal monitor program to run the generated program.

```

//jobname JOB USER=userid,.....
//stepname EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=4M
//STEPLIB DD DSN=DSN.SDSNLOAD,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
// DD DSN=cghlq.env.LOAD,DISP=SHR
//ELAPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//ELASNAP DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//EZEPRINT DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//SYSABOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM (ssid)
RUN PROG (appl-name) PLAN (plan-name) -
LIB ('cghlq.env.LOAD')
END
/*
//SYSTSPRT DD SYSOUT=*
//* Application specific DD statements
//file-name-1 DD .....
//file-name-n DD .....

```

Figure 13. JCL for Main Basic Program Run as z/OS Batch with DB2 Access

If the program calls a CICS program, include the following DD statement in the STEPLIB:

```
// DD DSN=CICSTS.V3R1M0.CICS.SDFHEXCI,DISP=SHR
```

Running Main Basic Program with DL/I Access

If a main basic program runs as a DL/I batch program, then all DL/I requests are handled by a private IMS region. The JCL for the step that runs the program must include DD statements for the IMS log if databases are opened with update intent or if the program uses the EGL `sysLib.audit()` system function. Also, a DD statement must be included for each of the data sets associated with the DL/I databases referenced in the IMS PSB. The IMS log DD statements (IEFRDER and IEFRDER2) are normally included in the DLIBATCH procedure.

EGL COBOL generation uses the `fda2mdli` JCL template to build the DD statements for program databases. This template has the DD statement commented out because EGL does not collect the high-level program database qualifiers. You need to provide the final tailoring of these DD statements in the sample runtime JCL. Alternatively, depending on your naming conventions, your administrator might be able to modify the `fda2mdli` template so that you can use the `symbolicParameter` build descriptor option to set high-level qualifiers for databases. Refer to the *EGL Generation Guide* in the EGL help system for information about modifying templates and using the `symbolicParameter` build descriptor option.

Figure 14 on page 104 shows the sample JCL used to run a generated program as a DL/I batch program.

```

//jobname JOB .....
//stepname EXEC DLIBATCH,DBRC=Y,
//          MBR=appl-name,PSB=ims-psb-name,BKO=Y,IRLM=N
//G.STEPLIB DD
//          DD
//          DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
//          DD DSN=cgh1q.env.LOAD,DISP=SHR
//* DFSVSAMP IS REQUIRED IF VSAM DATABASES - REPLACE MEMBER WITH
//* ONE THAT HAS VALID BUFFER POOL SIZES FOR YOUR APPLICATION
//G.DFSVSAMP DD DSN=ELA.V6R0M1;.ELASAMP(ELAVSAMP),DISP=SHR
//G.ELAPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//G.ELASNAP DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.EZEPRINT DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.SYSABOUT DD SYSOUT=*
//G.SYSOUT DD SYSOUT=*
//* Application specific DD statements including DL/I DB DD statements
//file-name-1 DD .....
//file-name-n DD .....

```

Figure 14. JCL for Main Basic Program Run as z/OS Batch with DL/I Access

Running a Main Basic Program with DB2 and DL/I Access

Figure 15 on page 105 shows the JCL that enables a program to run as a stand-alone DL/I batch processing program and to gain access to DB2 databases. Special recovery considerations are required. Refer to the DB2 documentation for your system for additional information.

The JCL for the step that runs the program must include DD statements for the IMS log if databases are opened with update intent or if the program uses the EGL **sysLib.audit()** system function. Also, a DD statement must be included for each of the data sets associated with the DL/I databases referenced in the IMS PSB. The IMS log DD statements (IEFRDER and IEFRDER2) are normally included in the DLIBATCH procedure.

EGL COBOL generation uses the JCL template fda2mdli to build the DD statements for DL/I program databases. This template has the DD statement commented out because EGL does not collect the high-level program database qualifiers. You need to provide the final tailoring of these DD statements in the sample runtime JCL. Alternatively, depending on your naming conventions, your administrator might be able to modify the fda2mdli template so that you can use the **symbolicParameter** build descriptor option to set high-level qualifiers for databases. Refer to the *EGL Generation Guide* for information about modifying templates and using the **symbolicParameter** build descriptor option.

```

//jobname JOB .....
//stepname EXEC DLIBATCH,DBRC=Y,
//          MBR=DSNMTV01,PSB=ims-psb-name,BKO=Y,IRLM=N
//G.STEPLIB DD
//          DD
//          DD DSN=DSN.SDSNLOAD,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
//          DD DSN=cgh1q.env.LOAD,DISP=SHR
/* DFSVSAMP IS REQUIRED IF VSAM DATABASES - REPLACE MEMBER WITH
/* ONE THAT HAS VALID BUFFER POOL SIZES FOR YOUR APPLICATION
//G.DFSVSAMP DD DSN=ELA.V6R0M1;.ELASAMP(ELAVSAMP),DISP=SHR
/*
//G.DDOTV02 DD DSN=&&TEMP1,
//          DISP=(NEW,PASS,DELETE),
//          SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA,
//          DCB=(RECFM=VB,BLKSIZE=4096,LRECL=4092)
//G.DDITV02 DD *
//          ssid,SYS1,DSNMIN10,,R,-,connection name,plan-name,appl-name
/*
//G.ELAPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//G.ELASNAP DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.EZEPRINT DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.SYSABOUT DD SYSOUT=*
//G.SYSOUT DD SYSOUT=*
/* Application specific DD statements including DL/I DB DD statements
//file-name-1 DD .....
//file-name-2 DD .....
/*
/* Attempt to print out the DDOTV02 data set created in previous step
//stepnam2 EXEC PGM=DFSERA10,COND=EVEN
//STEPLIB DD DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSNAME=&&TEMP1,
//          DISP=(OLD,DELETE)
//SYSIN DD *
CONTROL CNTL K=000,H=8000
OPTION PRINT
/*

```

Figure 15. JCL for Main Basic Program Run as z/OS Batch with DB2 and DL/I Access

Recovery and Restart for z/OS Batch Programs

For z/OS batch programs that use DL/I, the generated sample runtime JCL includes the parameter BKO=Y. If the program updates databases or files, specify BKO=Y in the runtime JCL in order to have rollback (ROLB) requests honored. If you specify BKO=N, DL/I returns status code AL for the roll-back call. Rational COBOL Runtime treats the AL status code as a soft error. No error message is issued, and processing continues.

You should develop recovery procedures in the event of program or system errors. Rational COBOL does not generate JCL to perform restart or recovery procedures.

Chapter 14. Preparing and Running Generated Programs in IMS/VS and IMS BMP

This chapter describes the steps required to prepare and run a generated COBOL program in an IMS environment:

- Modify the IMS system definition parameters
- Create the MFS control blocks
- Precompile, compile, link, and bind the generated program
- Make the new modules and MFS control blocks available to IMS
- Create or modify runtime JCL (IMS BMP only)

For general information on preparing programs for the runtime environment, see Chapter 9, “Output of Program Generation on z/OS Systems,” on page 71. For information about modifying JCL, see Chapter 12, “Creating or Modifying Runtime JCL on z/OS Systems,” on page 97.

Modifying the IMS System Definition Parameters

The following information describes the basic IMS system definition parameters that are required to run EGL-generated programs. You should review the performance options described in the IMS documentation for your system to determine the most effective options.

An IMS TRANSACT macro is required for each transaction code used to start an EGL main program in the IMS/VS environment and for each transaction-oriented BMP program. This includes the following transactions:

- Started from a clear IMS screen
- Used as a **sysVar.transactionID**
- Used as the target of a **transfer to transaction, show, or vgLib.startTransaction()** statement
- Transferred to by a non-EGL program
- Started as the result of an **add** statement that adds a transaction to a message queue
- Started by other IMS facilities

The TRANSACT macro must follow the APPLCTN macro for the IMS PSB that is to be used for the transaction.

Defining an Interactive Program

Each main transaction program must be defined as either an IMS message processing program (MPP) or a fast-path program with an associated transaction code, except when the program is started through a **transfer** statement of the form *transfer to a program* from another program.

Figure 16 on page 108 shows the system definition parameters that are required for defining an interactive EGL program.

APPLCTN	PGMTYPE=TP,PSB=ims-psb-name.	1
TRANSACT	CODE=trancode,	X2
	INQUIRY=NO,	X3
	MODE=SNGL,	X
	MSGTYPE=(SNGLSEG,RESPONSE),	X4
	EDIT=ULC,	X5
	SPA=(size,[DASD CORE],[FIXED])	6

Figure 16. IMS System Definition for an Interactive Transaction

- 1 The IMS PSB name and the EGL program name must match.
- 2 Multiple transactions can be associated with one program. If the program changes the value of **sysVar.transactionID** before a **converse**, include a TRANSACT macro for the original transaction code and a TRANSACT macro for the **sysVar.transactionID** value.
- 3 INQUIRY=NO is the default for IMS. If DL/I is used for the work database, INQUIRY=NO is required. The Rational COBOL Runtime work database supports help forms and displays data again if an input error occurs, as well as the **converse** statement. Therefore, even if the program databases are inquiry only, INQUIRY=NO is necessary. If DB2 is used for the work database and the program's use of all DL/I databases is inquiry only, then INQUIRY=YES can be used.
- 4 SNGLSEG is required. Either RESPONSE or NONRESPONSE can be used with Rational COBOL Runtime, depending on whether you want the keyboard to remain locked until the transaction completes. Even if NONRESPONSE mode is used, multiple simultaneous transactions from a single terminal are not supported.
- 5 Required for input in lowercase.
- 6 Include this parameter only if an IMS scratch pad area (SPA) is required. The SPA size is the length of the IMS SPA header (14 bytes) plus the length of the longest working storage record that might be received or sent during a **transfer to transaction** or **show** statement. However, if you include the **spaStatusBytePosition** and omit the **spaADF** build descriptor options, then you must add an additional byte when calculating the size. The SPA size must match the number specified for the **spaSize** build descriptor option when the program is generated.

You can also include the FPATH=YES parameter on the TRANSACT macro if the program might be run in an IMS Fast Path (IFP) region. If you include FPATH=YES, be sure to include the **imsFastPath="YES"** build descriptor option when you generate the program. Refer to the IMS manuals for your system for additional information about using IFP regions.

Defining Parameters for a Main Basic Program as an MPP

An EGL main basic program can also run as an asynchronous MPP. For example, an EGL main basic program can be used to process the information inserted to the message queue by a **sysLib.startTransaction()** statement or an **add** statement in another program. This type of program differs from one that runs as an IMS BMP in that the MPP cannot access any GSAM, indexed, or relative files, and cannot include any special restart logic. Figure 17 on page 109 shows the system definition parameters required for this case.

```

APPLCTN  PGMTYPE=TP,PSB=ims-psb-name           1
TRANSACT CODE=trancode,                         X   2
          MODE=SNGL

```

Figure 17. IMS System Definition for an Asynchronous MPP Program

- 1 The IMS PSB name and the EGL program name must match.
- 2 Multiple transactions can be associated with one program.

You can also include the `FPATH=YES` parameter on the `TRANSACT` macro if the program might be run in an IMS Fast Path (IFP) region. If you include `FPATH=YES`, be sure to include the `imsFastPath=YES` build descriptor option when you generate the program. Refer to the IMS manuals for your system for additional information about using IFP regions.

Defining Parameters for a Batch-Oriented BMP Program

If an EGL main basic program is generated to run as an IMS BMP program and it does not process an input message queue, it is a batch-oriented BMP program. Figure 18 shows the system definition parameters required for defining a main basic program as a batch-oriented BMP program.

```

APPLCTN  PGMTYPE=BATCH,PSB=ims-psb-name

```

Figure 18. IMS System Definition for a Main Basic Program Running as a Batch-Oriented BMP Program

Defining Parameters for a Transaction-Oriented BMP Program

If an EGL main basic program is generated to run as an IMS BMP program and it processes an input message queue created by MPP programs or by other BMP programs, it is a transaction-oriented BMP program. Figure 19 shows the system definition parameters that are required to define a main basic program as a transaction-oriented BMP program.

```

APPLCTN  PGMTYPE=BATCH,PSB=ims-psb-name
TRANSACT CODE=trancode,                         X   1
          MODE=SNGL,                             X
          WFI                                     2

```

Figure 19. IMS System Definition for a Main Basic Program Running as a Transaction-Oriented BMP Program

- 1 Multiple transactions can be associated with one program.
- 2 Wait-for-input (WFI) is optional. If it is specified, the program remains resident until the operator stops the transaction or shuts down the region.

Creating MFS Control Blocks

EGL generates message format services (MFS) control blocks when a FormGroup is generated for the IMS environment. The build script `FDAMFS` is used. `FDAMFS` has functionality similar to that of the `MFSUTL` and the `MFSTEST` JCL procedures that ship with the IMS product. When you generate the FormGroup, you specify the `mfsUseTestLibrary` build descriptor option to choose between the functionality of `MFSUTL` and `MFSTEST`. YES indicates `MFSTEST`.

When you set `mfsUseTestLibrary` to YES, the variable `MFSTEST` is set to YES in the build plan. The build script `FDAMFS` uses this variable to determine which of the JCL procedures (`MFSUTL` or `MFSTEST`) to follow. Refer to the message format

services documentation for your system for additional information about the MFS control blocks. Refer to the *EGL Generation Guide* for more information about the build descriptor options that control what is included in the MFS source.

If your program contains DBCS or mixed data, note that a long mixed constant field that results in multiple lines of MFS source might contain unpaired shift-in and shift-out characters. This occurs when the DBCS portion of the constant is split into more than one line. The MFS still works correctly.

Making New Modules Available in the IMS Environment

Whenever you install a new version of a program, MFS print services program, FormGroup format module, or DataTable, you need to recycle the message region.

If you generated with `mfsUseTestLibrary="YES"`, then the MFS control blocks were placed in the MFS test library (the TFORMAT library). To use the new version of the MFS control blocks, use the `/TEST MFS` command after you have signed on your IMS system and before you attempt to run a transaction that uses the new version of the forms.

If you generated with `mfsUseTestLibrary="NO"`, then the MFS control blocks were placed in the MFS staging library (FORMAT library). To use the new version of the MFS control blocks, you must do the following:

1. Run the IMS online change utility (OLCUTL) to copy the new MFS control blocks into the inactive format library.
2. Use the following IMS commands:

```
/MODIFY PREPARE FMTLIB  
/MODIFY COMMIT
```

Note: If the MFS control blocks and the FormGroup format module do not have the same generation date and time, Rational COBOL Runtime issues an error message.

Preloading Program, Print Services, and DataTable Modules

Preloading programs, MFS print services programs, FormGroup format modules, and DataTable modules that are frequently used might reduce the overhead of searching the STEPLIB, JOBLIB, link pack area, and link list. However, if modules are preloaded, they occupy virtual storage when they are not in use.

To improve response time, you might also preload modules associated with any transaction that might require better performance, even though the module itself is not frequently used.

To preload a program, MFS print services program, FormGroup format module, or DataTable program, have your system administrator do the following:

1. Put the module in a LNKLIST library.
2. Include the module name in a preload member (DFSMPLxx, where xx is a two-character ID that you select) in IMSVS.PROCLIB.
3. Indicate in the JCL for the IMS message region that the preload member is to be included.

For general information on preloading modules, see the IMS manuals for your system.

Running Programs under IMS

Prior to starting a generated program, the program user might be required to sign on to the IMS environment with a `/SIGN` command. Refer to the IMS documentation for information about the `/SIGN` command.

Starting a Main Program Directly

The simplest way for a program user to start an EGL program is by entering the IMS transaction code from an unformatted screen. The transaction code can be up to 8 characters. It is associated with the program in the IMS system definition `TRANSACT` macro. The following is an example of starting a transaction:

```
MYTRANS
```

IMS requires the transaction code to be followed by at least one blank prior to pressing the ENTER key.

Starting a Main Transaction Program Using the `/FORMAT` Command

A program user can use the IMS `/FORMAT` command to display a formatted screen to start a transaction if the `inputForm` specified for a program is defined with the IMS transaction code for the program as an 8-byte constant with the `protect=protect` and `intensity=invisible` properties. The attribute byte on the form becomes the attribute byte in the generated MFS. The 8-byte constant contains the name of the IMS transaction that is started when the form is processed.

The `/FORMAT` command directs IMS to display a screen format; however, the command does not cause the program to be run. After the program user enters data and presses the Enter key (or a function key), the message from the terminal is sent to the generated program for processing.

The syntax of the `/FORMAT` command is as follows:

```
/FORMAT modname [formName]
      or
/FOR modname [formName]
```

The *modname* operand is the FormGroup name (or alias name, up to a maximum of 6 characters) with an *O* suffix. The *formName* operand is required if there is more than one form in the FormGroup. It must be the form name that was specified as the `inputForm` for the program.

Because the transaction code must be included in the form, and a transaction code can only be associated with one program in the IMS system definition, only one program using the form can be started using the `/FORMAT` command.

Running Transaction Programs as IMS MPPs

Running generated programs is similar to running non-EGL-generated programs in the IMS MPP environment, with the following differences:

IMS Commands

The `/HOLD` command should be avoided. Rational COBOL Runtime uses the logical terminal identifier as the key of the work database. The data in the work database is destroyed if another generated program is run from the same terminal prior to resuming the original conversation.

Keyboard Key Operation

When the Clear key is pressed in IMS, IMS clears the screen, but does not notify the program. No transaction is scheduled, so the form is not automatically displayed again. If the program is conversational, the program user can enter the IMS /HOLD command followed immediately by an IMS /RELEASE command to display the form again.

When the EOF key is pressed in the first position of a field on a form, the data is not blanked. To blank the data, the program user must enter at least one blank before pressing the EOF key. Also, the program user should not use the DELETE CHARACTER key to erase the entire field because this is equivalent to pressing the EOF key in the first position of the field.

When typing over characters in a right-justified numeric field, any intervening spaces between the new digits entered and the original digits in the field should be deleted by pressing the DELETE CHARACTER key. Alternatively, the program user can type in all the digits for the new value and then use the EOF key to erase any remaining digits.

DBCS Data on a Non-DBCS Terminal

If a program inadvertently attempts to display a form with DBCS or mixed data on a non-DBCS terminal or printer, the results are unpredictable. The terminal might be logged off IMS and returned to the VTAM[®] sign-on screen without displaying any warning or error messages. If this happens, review your use of DBCS. Also, review your values for the **mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore** build descriptor options, and compare them to the IMS system definition for the terminal that had the problem.

Error Reporting

In certain error situations, Rational COBOL Runtime displays its own panel to explain the error to the program user. This occurs in the following situations:

- A message needs to be displayed, but the **msgField** property is not specified for the form. Form ELAM01 in FormGroup ELAxxx, where xxx is the national language code, is used.
- An unexpected program error has occurred. Form ELAM02 and (if necessary) continuation form ELAM03 are used to display the error messages. See “Using the Rational COBOL Runtime Error Panel” on page 144 for an example of ELAM02.

If an error occurs information might have been written to the message queue identified by the **errorDestination** build descriptor option for the first program in the run unit. See “IMS Diagnostic Message Print Utility” on page 135 for information on printing diagnostic errors.

Responding to IMS Error Messages

If a DFS message is displayed on your screen, make a note of the message. Then, depending on how your IMS system is set up, press either PA1 or PA2 to see if Rational COBOL Runtime has queued an error form to the terminal with more information. This can happen in the following situations:

- If Rational COBOL Runtime issues a ROLL call because of a run unit or catastrophic error, IMS issues the message:

```
DFS555I TRAN tttttttt ABEND S000,U0778 ; MSG IN PROCESS:
ttttttt mmmmmmmmmMAP ;;;;gdate gtime rdate rtime
```

Where *ttttttt* is the IMS transaction code, *mmmmmmmm* is the form name, *gdate* and *gtime* are the date and time the FormGroup was generated, and *rdate* and *rtime* are the date and time of the abend.

The DFS555I message is also used by IMS when other abends occur, including the 1600, 1601, 1602, and 1606 abends from Rational COBOL Runtime.

- If Rational COBOL Runtime ends the run unit for a transaction program that was generated with **imsFastPath="YES"** and is being run in an IMS fast-path region, IMS issues the message:
DFS2766I PROCESS FAILED
- If Rational COBOL Runtime abnormally ends the logical unit of work (LUW) for a transaction program that was generated with **imsFastPath="YES"**, IMS might issue the message:
DFS2082I RESPONSE MODE TRANSACTION TERMINATED WITHOUT REPLY

See Chapter 18, “Diagnosing Problems for Rational COBOL Runtime on z/OS Systems,” on page 139 for information on diagnosing errors.

Running Main Basic Programs as MPPs

An EGL main basic program can be generated to run in the IMS MPP environment. In this situation, IMS automatically starts the transaction whenever a message is written to the message queue associated with the transaction.

If an error occurs information might have been written to the message queue identified by the **errorDestination** build descriptor option for the first program in the run unit. See “IMS Diagnostic Message Print Utility” on page 135 for information on printing diagnostic errors.

Running a Main Basic Program under IMS BMP

A main basic program generated for the IMS BMP environment can be started by submitting JCL. Called programs can only be started by another EGL program or by a non-EGL program.

The EGL COBOL generation process creates sample runtime JCL for running programs in the IMS BMP environment. The generated JCL has the same name as the program. If you set the **genRunFile** build descriptor option to YES, sample JCL is created specifically for the program during program generation. The build plan uploads the sample runtime JCL to a z/OS partitioned data set (PDS).

The JCL might need to be modified to add data sets required by called or transferred-to programs. You also need to modify the JCL to add any data sets that are dynamically allocated with the **recordName.resourceAssociation** or **converseVar.printerAssociation** system variables. See Chapter 12, “Creating or Modifying Runtime JCL on z/OS Systems,” on page 97 for more information on modifying the sample runtime JCL.

If you get a JCL error for the runtime JCL, check the Generation Results view for the programs involved for any error messages related to JCL generation. In addition, ensure the tailoring that was done for the JCL templates is correct. Also check any changes you made when you customized the sample runtime JCL.

The following sections show JCL for different IMS BMP programs.

Examples of Runtime JCL for IMS BMP Programs

The generated JCL in the following examples has these characteristics:

- The examples are based on the JCL templates shipped with EGL. Your actual JCL templates might differ if your system administrator has tailored them for your organization. Refer to the *EGL Generation Guide* for more information about tailoring JCL templates.
- Lowercase text appears in the examples where a generic example name has been substituted for an actual program or data set name.
- EZEPRINT is always routed to SYSOUT=*

If you route EZEPRINT to a data set, you must use the following DCB attributes:

- LRECL=137, BLKSIZE=141, RECFM=VBA if the FormGroup does not contain any DBCS forms
- LRECL=654, BLKSIZE=658, RECFM=VBA if the FormGroup contains any DBCS forms

You cannot use FormGroups that do not have any DBCS forms with FormGroups that do have DBCS forms in a single job step.

The first library in the STEPLIB concatenation sequence must have the largest block size, or BLKSIZE=32760 can be specified on the first STEPLIB DD statement for the step.

Running a Main Basic Program as an IMS BMP Program

If a main basic program runs as an IMS BMP program, all DL/I requests are passed to a central copy of IMS which coordinates updates to the databases across multiple BMPs and MPPs. The DD statements for the IMS log and the program databases are not required in the JCL for the BMP job step. These databases and the IMS log are allocated to the IMS control region.

Figure 20 shows a sample set of JCL to run a generated program as a BMP program.

```
//jobname JOB .....
//stepname EXEC IMSBATCH,
//   MBR=appl-name,PSB=ims-psb-name,IN=trans-name
//G.STEPLIB DD
//   DD
//   DD DSN=CEE.SCEERUN,DISP=SHR
//   DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
//   DD DSN=cghlq.env.LOAD,DISP=SHR
//G.ELAPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//G.ELASNAP DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.EZEPRINT DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.SYSABOUT DD SYSOUT=*
//G.SYSOUT DD SYSOUT=*
/* Application specific DD statements for files
/* No application specific DD statements for databases required
//file-name-1 DD .....
//file-name-n DD .....
```

Figure 20. JCL for Main Basic Program as an IMS BMP Program

If you run a transaction-oriented BMP program, the *trans-name* must be set to the name of the transaction for the message queue that the BMP program processes. If not, *trans-name* should be a null value. The sample runtime JCL created by EGL defaults *trans-name* to the program name for a transaction-oriented BMP program

that uses **get next** to read the message queue. The sample runtime JCL created by EGL defaults *trans-name* to null for batch-oriented BMP programs or for transaction-oriented BMP programs that use **VGLib.VGTDLI()**, **dliLib.AIBTDLI()**, or **dliLib.EGLTDLI()** to read the message queue.

If the BMP program uses GSAM, the following DD statements are also included in the sample runtime JCL:

```
//IMS      DD DSN=IMS.PSBLIB,DISP=SHR
//          DD DSN=IMS.DBDLIB,DISP=SHR
```

These DD statements are generated from the fda2mims JCL template.

Running a Main Basic Program as an IMS BMP Program with DB2 Access

Figure 21 shows a sample set of JCL to run a generated program that accesses DB2 resources as a BMP. The DD statements for the IMS log and the DL/I program databases are not required in the JCL for the BMP job step. The DL/I databases and the IMS log are allocated to the IMS control region.

```
//jobname JOB .....
//stepname EXEC IMSBATCH,
dliLib.//   MBR=appl-name,PSB=ims-psb-name,IN=trans-name
//G.STEPLIB DD
//          DD
//          DD DSN=DSN.SDSNLOAD,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR
//          DD DSN=cghlq.env.LOAD,DISP=SHR
//G.DFSESL  DD DSN=IMS.RESLIB,DISP=SHR
//          DD DSN=DSN.SDSNLOAD,DISP=SHR
//G.ELAPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//G.ELASNAP DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.EZEPRINT DD SYSOUT=*,DCB=(RECFM=VBA,BLKSIZE=4096)
//G.SYSABOUT DD SYSOUT=*
//G.SYSOUT  DD SYSOUT=*
//* Application specific DD statements for files
//* No application specific DD statements for databases required
//file-name-1 DD .....
//file-name-n DD .....
```

Figure 21. JCL for Main Basic Program as an IMS BMP Program with DB2

If you run a transaction-oriented BMP program, the *trans-name* must be set to the name of the transaction for the message queue that the BMP program processes. If not, *trans-name* should be a null value. The sample runtime JCL created by EGL defaults *trans-name* to the program name for a transaction-oriented BMP program that uses **get next** to read the message queue. The sample runtime JCL created by EGL defaults *trans-name* to null for batch-oriented BMP programs or for transaction-oriented BMP programs that use **VGLib.VGTDLI()**, **dliLib.AIBTDLI()**, or **dliLib.EGLTDLI()** to read the message queue.

If the BMP program uses GSAM, the following DD statements are also included in the sample runtime JCL:

```
//IMS      DD DSN=IMS.PSBLIB,DISP=SHR
//          DD DSN=IMS.DBDLIB,DISP=SHR
```

These DD statements are generated from the fda2mims JCL template.

Recovery and Restart for IMS BMP Programs

You should develop recovery procedures in the event of program or system error. Rational COBOL does not generate JCL to perform restart or recovery procedures.

If your IMS BMP program ends with a run unit or catastrophic error, all updates after the last checkpoint are rolled back and the program ends. You should include checkpoint and restart logic in the program if it is to run as an IMS BMP. Refer to the IMS documentation for your system for additional information about checkpoint and restart.

Chapter 15. Moving Prepared Programs to Other Systems from z/OS Systems

You might need to move a prepared program from one system to another. For example you might have the compiler on one host development machine but want to run the program on several production machines.

If you use DB2, the DB2 BIND must be done on the production system.

The COBOL and Rational COBOL Runtime products on the production machine must be at the same maintenance level as, or a higher level than, on the development machine.

Moving Prepared Programs To Another z/OS System

If a program has been completely prepared on one system and you want to move the prepared program to another system, perform the following steps:

1. Copy the program-related parts (including the FormGroup and DataTable parts) to the production system. The names of the source libraries are shown with the default naming convention used in the build scripts, where *cghlq* is the user or project-related high level qualifier and *env* is the runtime environment code.

Table 20. Parts to Copy

Data Set Name	Contents
cghlq.env.LOAD	Program, library, service, print services program, FormGroup format modules, and DataTable modules.
cghlq.env.DBRMLIB	DB2 database request modules (DBRMs) for SQL programs
cghlq.env.EZEBIND	BIND commands for SQL programs
cghlq.env.EZEMFS	MFS source for IMS/VS and IMS BMP FormGroups
cghlq.env.EZEJCLX	Runtime JCL for IMS BMP and z/OS batch programs

Note:

The *cghlq* variable comes from the **projectID** build descriptor option. The *env* variable comes from the **system** build descriptor option.

2. Provide your own JCL to build the plans for DB2 programs using the BIND commands from the BIND library and the DBRMs from the DBRM library. You need to edit the EZEBIND member, and make the appropriate changes such as DB2 subsystem name or collection IDs to match the new system where you are moving the program.
3. Provide your own JCL to assemble the MFS control blocks for IMS/VS and IMS BMP. It is much easier to assemble the MFS source on the production system than to try to locate the DIF/DOF and MID/MOD in the MFS format libraries. However, if you have procedures in place to move the DIF/DOF and MID/MOD to a different system, you can use these procedures instead of moving the MFS source in the EZEMFS library.

4. Follow the procedures identified in this manual for defining programs to CICS or IMS.
5. Define files and databases used by the program on the new system.

Maintaining Backup Copies of Production Libraries

Follow your installation-defined guidelines and procedures for making backup copies of production libraries. Having backup copies of production libraries enables you to return to the prior level of a program in case of errors. The production libraries for which copies should be made are those listed in Table 20 on page 117.

Part 4. Utilities

Chapter 16. Using Rational COBOL Runtime

Utilities for z/OS CICS Systems	121
Using the CICS Utilities Menu.	121
New Copy	122
Diagnostic Message Printing Utility	124
Diagnostic Control Options for z/OS CICS Systems	125
Change or View Diagnostic Control Options for a Transaction	126
Change or View Default Diagnostic Control Options	128
Using the Parameter Group Utility for z/OS CICS Systems	129

Chapter 17. Using Rational COBOL Runtime

Utilities for IMS Systems	135
IMS Diagnostic Message Print Utility	135

Chapter 16. Using Rational COBOL Runtime Utilities for z/OS CICS Systems

Rational COBOL Runtime provides a set of utilities in CICS to help manage the error diagnosis and control facilities of the Rational COBOL Runtime runtime environment. You can access these utilities from the CICS utilities menu.

Using the CICS Utilities Menu

To access the CICS utilities do the following:

1. Log on to CICS.
2. Type ELAM on a clear screen.
3. Press Enter. When the ELAM transaction is started, a copyright panel is displayed.
4. Press Enter. The CICS Utilities Menu (Figure 22) is displayed.

```
ELAM                                Rational COBOL Runtime
                                CICS Utilities Menu

Select one of the following utilities; then press Enter.

Action...._

_1. New Copy
_2. Diagnostic Message Printing
_3. Diagnostic Control Options

ENTER  F1=HELP  F3=EXIT
```

Figure 22. CICS Utilities Menu

Three functions are available from the CICS Utilities Menu panel (Figure 22):

New Copy

This function causes a new copy of a program, FormGroup, or DataTable to be used by subsequent transactions. Use the new copy function when programs, libraries, services, FormGroups, and DataTables are modified and generated again.

For programs, libraries, services, and FormGroups, you can use either the Rational COBOL Runtime new copy utility or the CICS NEWCOPY command to cause the new copy of the program to be used the next time a load request is issued for the program.

The Rational COBOL Runtime new copy utility does a new copy for both the online print services program and the FormGroup format module when you specify a part type of FormGroup. If you use the CICS NEWCOPY command for a FormGroup, you must issue the NEWCOPY for both the online print services program and the FormGroup format module.

For a DataTable, you must use the Rational COBOL Runtime new copy utility to cause a fresh copy of the DataTable to be used the next time a load request is issued for the DataTable. Do not use the CICS NEWCOPY command for DataTables.

Diagnostic Message Printing

This function routes the diagnostic messages in an error destination transient data queue to a spool file for printing or subsequent processing.

Diagnostic Control Options

This function lets you view or change the diagnostic control options set for the installation or for individual transactions. The options include dump control, error message routing to a transient data queue or the CICS journal, and transaction disabling when serious problems occur.

New Copy

The Rational COBOL Runtime new copy utility causes a new copy of a program, FormGroup, or DataTable to be used by subsequent transactions. Transactions that are in progress when this function was started continue to use the copy that was current when the transaction began. Programs must end or reach a segment break before the new copy is used.

The Rational COBOL Runtime new copy utility must be run separately for programs, libraries, services, FormGroups, and DataTables to replace the copy already in storage.

To gain access to the Rational COBOL Runtime new copy utility, do the following:

1. Select option 1, New Copy, on the CICS Utilities Menu panel (Figure 22).
2. Press Enter.

The New Copy panel (Figure 23 on page 123) is displayed.

Note: You can also gain access to the Rational COBOL Runtime new copy utility by doing the following:

1. Type ELAN on a clear screen.
2. Press Enter. When the ELAN transaction is started, a copyright panel is displayed.
3. Press Enter. The New Copy panel (Figure 23 on page 123) is displayed.

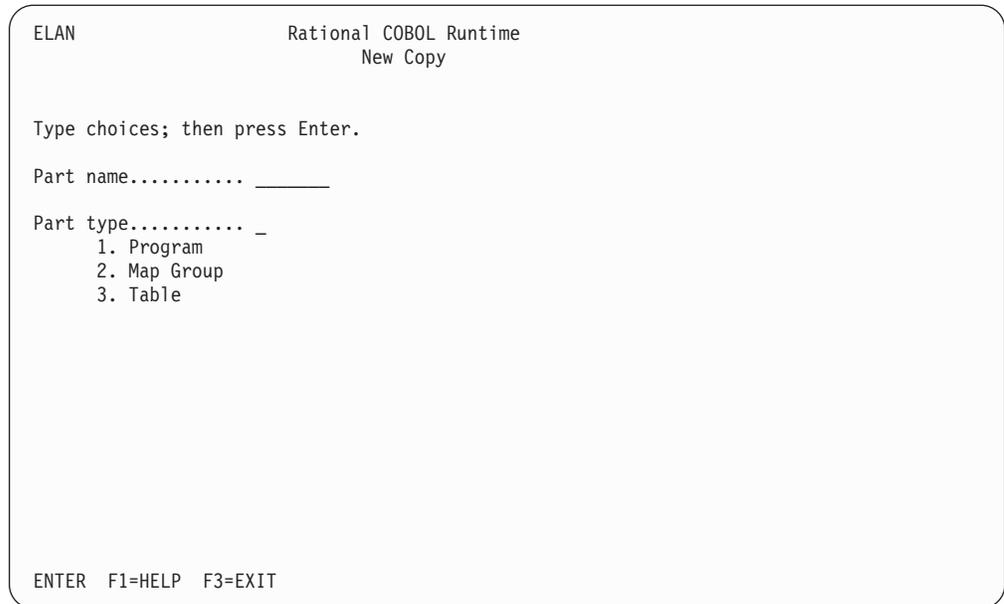


Figure 23. New Copy panel

Enter the following on the New Copy panel:

Part name

Specifies the name of the program, FormGroup, or DataTable to be used as a new copy in subsequent transactions

Part type

Specifies the type of part to be replaced

Note: Rational COBOL Runtime does not validate the part type. You must specify the correct type because different processing is required for programs, FormGroups, and DataTables. If you have problems in processing after using the Rational COBOL Runtime new copy utility, try the Rational COBOL Runtime new copy utility again to ensure you specified the part type correctly.

The correct type can be one of the following:

Program

This type causes the utility to issue a CICS SET PROGRAM(name) NEWCOPY command to access a new copy of the program, library, or service. This command does not cause a new copy for called programs that are statically linked with their caller.

Map Group

(EGL FormGroup) This type causes the utility to issue a CICS SET PROGRAM(name) NEWCOPY command to access a new copy of the FormGroup format module and the online print services program associated with the FormGroup.

Table

(EGL DataTable) This type causes the utility to issue a CICS SET PROGRAM(name) NEWCOPY command to access a new copy of the DataTable program and sets a flag for Rational COBOL Runtime, indicating that a new copy of the DataTable is to be used the next time a program loads the DataTable contents.

If the DataTable has been generated as a shared DataTable, currently running transactions continue to use the old copy of the DataTable while new transactions share the new copy of the DataTable.

You can also access the new copy utility in batch mode. To invoke the batch new copy utility, link to program ELABNEW:

```
EXEC CICS LINK PROGRAM("ELABNEW")
COMMAREA(passed-parms)
LENGTH(174)
```

where the **passed-parms** record has the following structure:

Field	Length in Bytes	Type of Data	Description
NLS code	3	Character	NLS code identifying the language
Part name	8	Character	Name of program, FormGroup, or DataTable to be used as a new copy in subsequent transactions
Part type	1	Character	Type of part to be replaced: "1" Program, Library, Service "2" FormGroup "3" DataTable For more information, press F1 to see the description for part type.
Return code	2	Binary	Return code from new copy
Message 1	80	Character	Message returned from new copy
Message 2	80	Character	Message returned from new copy

The following fields must be provided by the user:

- NLS code
- Part name
- Part type

The other fields are filled in by the new copy utility.

Any nonzero return code means that the new copy operation was not successful. If a nonzero value is returned in the **return code** field, check messages 1 and 2 for details indicating what error occurred.

Note: Message 2 is not always filled in. It may be blank.

Diagnostic Message Printing Utility

Diagnostic message printing allows you to route diagnostic messages in an error destination transient data queue to a JES spool file for printing.

To gain access to the diagnostic message print utility do the following:

1. Select option 2, Diagnostic Message Printing, from the CICS Utilities Menu panel (Figure 22 on page 121).
2. Press Enter.

The Diagnostic Message Printing panel (Figure 24 on page 125) is displayed.

Note: You can also access the diagnostic message print function by doing the following:

1. Type ELAU on a clear screen.
2. Press Enter. When ELAU is started, a copyright panel is displayed.
3. Press Enter. The Diagnostic Message Printing panel (Figure 24) is displayed.

```
ELAU                                Rational COBOL Runtime
                                     Diagnostic Message Printing

Fill in the appropriate fields; then press Enter.

Error destination queue name.....ELAD

JES Spool File Information

Node.....*
Userid.....*
Class.....A

Clear destination queue.....Y Y=Yes, N=No

ENTER  F1=HELP  F3=EXIT
```

Figure 24. Diagnostic Message Printing panel

You can enter information in the following fields on the Diagnostic Message Printing panel:

Error destination queue name

This field specifies the name of an existing error destination.

Enter the 1 to 4 character DCT name of the error destination transient data queue. The default is ELAD. You can either leave the messages in the queue or clear them after they have been printed.

JES Spool File Information

This field specifies the spool file where the messages are to be written. If you do not specify anything in these fields, the system uses the default values (shown in Figure 24) which route the report to the local spool printer for your CICS system.

Clear destination queue

This field specifies whether to clear the error queue of all messages after the messages are written to a spool file. The default is Y.

Diagnostic Control Options for z/OS CICS Systems

The diagnostic control options utility enables you to alter the diagnostic action options taken for a given transaction code that is assigned to a generated CICS program. If multiple transaction codes are assigned to a program, each transaction code is specified independently to the diagnostic control options utility.

You can also specify a default action to take place for transactions that are not explicitly defined to the diagnostic control options utility.

To gain access to the diagnostic control options utility, do the following:

1. Select option 3, Diagnostic Control Options, from the CICS Utilities Menu (Figure 22 on page 121).
2. Press Enter. The Diagnostic Control Options panel (Figure 25) is displayed.

Note: You can also gain access to the diagnostic control options utility by doing the following:

1. Type ELAC on a clear screen.
2. Press Enter. When ELAC is started, a copyright panel is displayed.
3. Press Enter. The Diagnostic Control Options panel (Figure 25) is displayed.

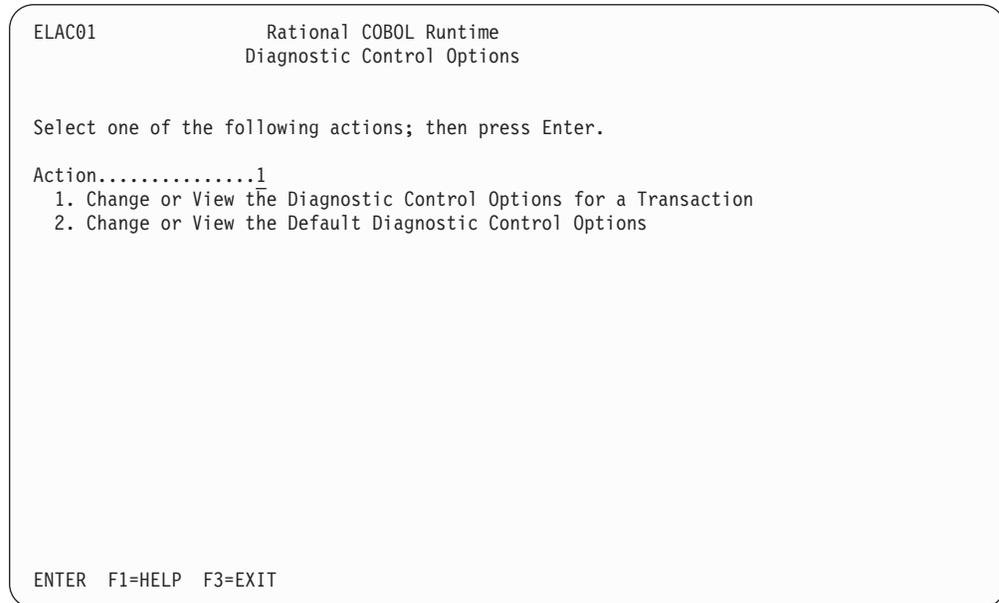


Figure 25. Diagnostic Control Options panel

You can access the following functions from the Diagnostic Control Options panel:

Change or View the Diagnostic Control Options for a Transaction

This option enables you to change or view the diagnostic options for a specific transaction code.

Change or View the Default Diagnostic Control Options

This option enables you to change or view the installation default diagnostic options.

This affects transaction codes that are not specifically identified to the diagnostic controller.

Change or View Diagnostic Control Options for a Transaction

This function enables you to change the Rational COBOL Runtime error diagnostic and control options in effect for a specific CICS transaction.

To start the function do the following:

1. Select option 1, Change or View the Diagnostic Control Options for a Transaction, from the Diagnostic Control Options panel (Figure 25).
2. Press Enter. The Change or View Diagnostic Control Options for a Transaction panel (Figure 26 on page 127) is displayed.

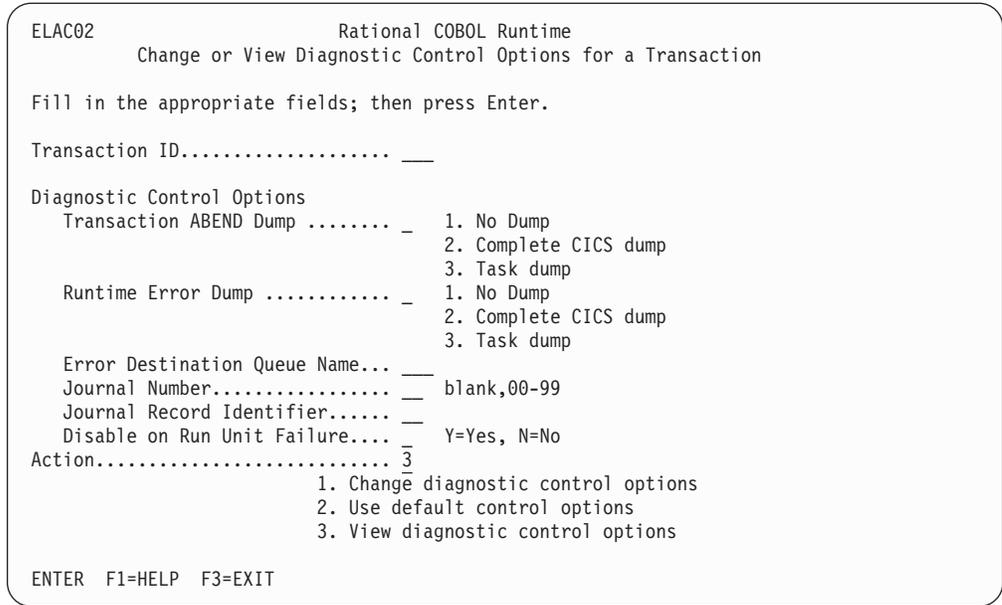


Figure 26. Change or View Diagnostic Control Options for a Transaction panel

The following fields can be entered on the Change or View Diagnostic Control Options for a Transaction panel :

Transaction ID

Specifies the 1 to 4 character identifier of the transaction you want to change the diagnostic options for

Diagnostic Control Options

Transaction ABEND Dump

Specifies the type of dump taken on a CICS transaction ABEND

The types of dumps are:

- 1. No Dump
- 2. Complete CICS dump
- 3. Task dump

Runtime Error Dump

Specifies the type of dump taken on a Rational COBOL Runtime-detected error for which a dump is indicated in the error message explanation

The types of dumps are:

- 1. No Dump
- 2. Complete CICS dump
- 3. Task dump

Error Destination Queue Name

Specifies the 1 to 4 character name of a transient data queue to which Rational COBOL Runtime error diagnostic messages are written whenever a transaction ends abnormally due to an error

If this field is blank, no messages are written to a queue.

Journal Number

Specifies the journal number of the CICS journal to which error diagnostic messages are written whenever a transaction is not successful due to an error

If this field is blank, no journal messages are written.

Journal Record Identifier

Specifies the 1 to 2 character record identifier used when messages are written to the CICS journal

If this field is blank, the default identifier EZ is used.

Disable on Run Unit Failure

Specifies whether a transaction is disabled whenever an error is detected that is likely to occur each time the transaction is run

Y Specifies that the transaction is disabled when these errors are detected

N Specifies that the transaction is not be disabled

Action

Allows you to change the current options, view the current options, or accept the default options

To change the options currently set for a transaction do the following:

1. Specify the transaction identifier and any changes.
2. Select 1, Change diagnostic control options.
3. Press Enter.

To use the installation defaults for the transaction do the following:

1. Type the transaction identifier.
2. Select 2, Use default control options.
3. Press Enter.

To view the options currently set for a transaction do the following:

1. Type the transaction identifier.
2. Select 3, View diagnostic control options.
3. Press Enter.

Change or View Default Diagnostic Control Options

This function enables you to change or view the default diagnostic options for transactions that are not identified to the diagnostic controller. If your default options were not modified at installation, the default diagnostic options are set as follows:

- Transaction ABEND and runtime errors both cause a task dump.
- The error destination queue name is ELAD.
- Diagnostic messages are not written to a CICS journal data set.
- Transactions are not disabled on a run unit error.

To start this function do the following:

1. Select 2, Change or View the Default Diagnostic Control Options, from the Diagnostic Control Options panel (Figure 25 on page 126).
2. Press Enter. The Change or View Default Diagnostic Control Options panel is displayed:

```

ELAC04                                Rational COBOL Runtime
                                Change or View Default Diagnostic Control Options

Fill in the appropriate fields; then press Enter.

Default Diagnostic Control Options

Transaction ABEND Dump ..... 3      1. No Dump
                                       2. Complete CICS dump
                                       3. Task dump

Runtime Error Dump ..... 3          1. No Dump
                                       2. Complete CICS dump
                                       3. Task dump

Error Destination Queue Name... ELAD
Journal Number..... blank,00-99
Journal Record Identifier..... EZ
Disable on Run Unit Failure.... N   Y=Yes, N=No

ENTER  F1=HELP  F3=EXIT

```

Figure 27. Change or View Default Diagnostic Control Options

The options on this panel are the same as those defined for changing or viewing the diagnostic control options for a transaction. They are all defined following Figure 26 on page 127.

Using the Parameter Group Utility for z/OS CICS Systems

Use the parameter group utility to create and maintain the parameter groups in the parameter group file. Each group contains parameters for controlling terminal printer utility (FZETPRT) transactions.

See “Special Parameter Group for the FZETPRT Program” on page 36 for a description of the startup parameters that can be included in the parameter group used with the FZETPRT program.

You can use the parameter group utility to perform the following operations:

- Display the contents of existing parameter groups
- View a list of existing parameter group names
- Add a new parameter group
- Change a parameter group
- Delete a parameter group

Table 21 on page 130 shows the steps used to define a parameter group file.

Table 21. Defining Parameter Group Files for z/OS CICS

Procedure

1. Define the parameter group file using the IDCAMS utility.
DEFINE CLUSTER (NAME(PARM.GROUP.FILE)-
RECORDS(100 100) KEYS(16 0) RECORDSIZE(272 272) INDEXED)
2. Initialize the parameter group file by using the IDCAMS REPRO function to insert a dummy record into the file.
3. Specify the FCT for the parameter group file utility to have access to a user-defined message file for CICS.

```
DFHFCT TYPE=DATASET,           C
      DATASET=EZEPRMG,         C
      ACCMETH=VSAM,           C
      SERVREQ=(READ,UPDATE,ADD,DELETE,BROWSE), C
      FILESTAT=(ENABLED,CLOSED), C
      RECFORM=FIXED           C
```

4. Allocate the file by adding the following statement to the z/OS CICS startup JCL:

```
//EZEPRMG DD DISP=SHR,DSN=PARM.GROUP.FILE
```

Note: The name that designates the parameter group file (EZEPRMG) is a reserved file name and cannot be used as a data file by an EGL-generated program.

When the file has been created and allocated, you can access the parameter group utility by doing the following:

1. Log on to CICS.
2. Type ELAP on a clear screen.
3. Press Enter.

The parameter group utility does not give message-specific tutorial help after a message is displayed and PF1 is pressed.

After the parameter group utility has been started, the Parameter Group Specification panel (Figure 28) is displayed. You can specify the parameter group name on this panel.

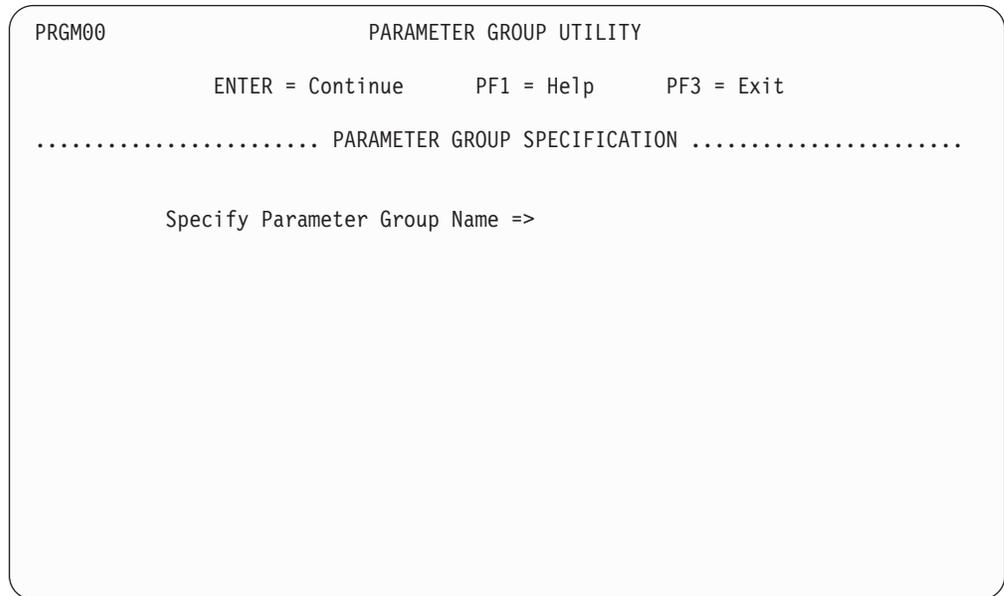


Figure 28. Parameter Group Specification panel

The parameter group name can be from 1 to 4 alphanumeric characters and must be the name of the transaction that was used to start the FZETPRT program. (The utility does not verify this.)

You can enter a group name that already exists if you want to modify a parameter group, or you can enter one that does not exist if you want to define a new parameter group.

Entering a question mark (?) as the group name on the Parameter Group Specification panel displays a list of previously-defined group names on the next panel, the Parameter Group List Display panel (Figure 29). Entering some characters followed by an asterisk (*) displays a list of parameter group names that begin with the characters that you entered. Entering a specific parameter group name displays the Parameter Group Definition panel (Figure 30 on page 132).

```

PRGM01                PARAMETER GROUP UTILITY

                ENTER = Continue  PF3 = Exit  PF4 = Refresh  PF1 = Help
                PF7 = Back        PF8 = Forward

..... PARAMETER GROUP LIST DISPLAY .....

___ PRIN            ___ USRQ

```

Figure 29. Parameter Group List Display panel

From the Parameter Group List Display panel, you can select a group name to edit by typing an S in the selection field to the left of the group name. You can delete a group by typing a D in the selection field to the left of the group name.

If the specified parameter group already exists, its contents are displayed on the Parameter Group Definition panel. The parameter group can be altered. If the specified parameter group does not exist, the Parameter Group Definition panel is displayed without any data. You can define the new contents; up to 256 characters of data can be entered for a parameter group.

```

PRGM02                PARAMETER GROUP UTILITY

                PA2 = Cancel  PF1 = Help  PF3 = File and Exit
                Parameter Group = CCCCCC

.....PARAMETER GROUP DEFINITION.....

Parameter Group:

=>PRTBUF=2048 PRTMPP=132 PRTTYP=D FORMFD=NO

```

Figure 30. Parameter Group Definition panel

The parameter group utility does not validate or format the parameters that are specified on the Parameter Group Definition panel. Any parameters that are not

valid are ignored when the FZETPRT program is started. For more information about setting the parameters for terminal printing, see “Special Parameter Group for the FZETPRT Program” on page 36.

If you press PF3 on the Parameter Group Definition panel without entering any parameters, a parameter group is stored without any associated parameters. You can store an empty parameter group to reserve parameter group names.

Empty parameter groups do not affect the initialization of the FZETPRT program.

The parameter group utility left-justifies the parameter group name and pads it to the right with blanks (X'40'). The parameter group utility uses this name as a key to index the parameter group file.

If you selected a parameter group from the Parameter Group List Display panel (Figure 29 on page 132), after the Parameter Group Definition panel is processed, the Parameter Group List Display panel is displayed again with the original request replaced by an asterisk beside the group name that was processed. An asterisk (*) is ignored as input on the Parameter Group Definition panel if more processing is done.

Chapter 17. Using Rational COBOL Runtime Utilities for IMS Systems

Rational COBOL Runtime provides a utility in IMS to print diagnostic information.

IMS Diagnostic Message Print Utility

When a generated program ends abnormally due to an error condition in IMS environments, diagnostic error messages are written to the message queue identified by the **errorDestination** build descriptor option for the first program in the run unit.

An IMS BMP program is provided to print the messages in the message queue. The JCL needed to print the diagnostic information is supplied as member ELAMQJUD of ELA.V6R0M1;ELAJCL (see Figure 31).

The message queue identified by the IN parameter is the name of the queue that was specified for **errorDestination** when the program was generated. The default name is ELADIAG.

```
//*****00000100
//** ELAMQJUD - JCL TO DRAIN AND PRINT THE ELADIAG MESSAGE QUEUE      00000200
//**          FOR IBM RATIONAL COBOL RUNTIME.                        00000300
//**          THIS PROGRAM RUNS AS A BMP.                            00000400
//**                                                                00000500
//** LICENSED MATERIALS - PROPERTY OF IBM                          00000600
//** 5655-R29 (C) COPYRIGHT IBM CORP. 2000, 2006                   00000700
//** SEE COPYRIGHT INSTRUCTIONS                                    00000800
//**                                                                00000900
//** STATUS = VERSION 6, RELEASE 0, LEVEL 1                        00001000
//**                                                                00001100
//** TO TAILOR THIS JOBSTREAM:                                     00001200
//**     1. COPY A JOBCARD.                                        00001300
//**     2. CHANGE IN= TO THE NAME OF YOUR ERROR DIAGNOSTIC      00001400
//**        QUEUE.                                              00001500
//**     3. MAKE SURE THAT THE TRANSACTION SPECIFIED BY IN=     00001600
//**        AND THE ELAMPUTL PROGRAM ARE STARTED BY IMS.        00001700
//**                                                                00001800
//** RETURN CODES                                               00001900
//**     0 - SUCCESSFUL COMPLETION                                00002000
//**     4 - NO MESSAGES ON QUEUE TO DRAIN.                      00002100
//**     16 - FATAL ERROR. PROCESSING TERMINATED                00002200
//**     20 - OPEN FAILED ON ELAPRINT                            00002300
//**                                                                00002400
//*****00002500
//DRAINMQ      EXEC IMSBATCH,MBR=ELAEPUTL,                          00002600
//             PSB=ELAMPUTL,IN=ELADIAG,RGN=4096K                  00002700
//G.STEPLIB    DD                                               00002800
//             DD                                               00002900
//             DD DSN=CEE.SCEERUN,DISP=SHR                        00003000
//             DD DSN=ELA.V6R0M1;.SELALMD,DISP=SHR                00003100
//G.ELAPRINT   DD SYSOUT=*                                        00003200
//G.SYSOUT     DD SYSOUT=*                                        00003300
//G.SYSPRINT   DD SYSOUT=*                                        00003400
/*                                                    00003500
```

Figure 31. ELAMQJUD

Part 5. Diagnosing Problems

Chapter 18. Diagnosing Problems for Rational COBOL Runtime on z/OS Systems 139

Detecting Errors	139
Reporting Errors	139
Controlling Error Reporting	140
Controlling Error Reporting in CICS.	140
Controlling Error Reporting in IMS Environments	140
Controlling Error Reporting in z/OS Batch	141
Error Reporting Summary	141
Transaction Error	141
Run Unit Error	142
Catastrophic error	143
Rational COBOL Runtime Error	144
Using the Rational COBOL Runtime Error Panel	144
Printing Diagnostic Information for IMS	145
errorDestination Message Queue	145
IMS Log Format	146
Running the Diagnostic Print Utility.	147
Printing Diagnostic Information for CICS	147
CICS Diagnostic Message Layout.	147
Running the Diagnostic Print Utility.	148
Analyzing Errors Detected while Running a Program	148

Chapter 19. Finding Information in Dumps.	151
Rational COBOL Runtime ABEND Dumps	151
COBOL or Subsystem ABEND Dumps	151
Information in the Rational COBOL Runtime Control Block	152
Information in a Program, Print Services, or DataTable Profile Block	152
How to Find the Current Position in a Program at Time of Error	153

Chapter 20. Rational COBOL Runtime Trace Facility	155
Enabling EGL Program Source-Level Tracing with Build Descriptor Options	155
Activating a Trace	156
Activating a Trace Session for CICS or IMS/VS	156
Activating a Trace Session for z/OS Batch or IMS BMP.	159
Deactivating a Trace Session	161
Printing Trace Output	161
Printing the Trace Output in CICS	161
Printing the Trace Output in IMS/VS	161
Printing the Trace Output in z/OS Batch or IMS BMP	161
Reporting Problems for Rational COBOL Runtime	161

Chapter 21. Common Messages during Preparation for z/OS Systems	163
Common Abend Codes during Preparation	163
MFS Generation Messages	163

DB2 Precompiler and Bind Messages	164
COBOL Compilation Messages	164

Chapter 22. Common System Error Codes for z/OS Systems.	167
Common Error Codes	167
System Error Code Formats for sysVar.errorCode	167
Common System Error Codes in sysVar.errorCode	170
EGL Error Codes	171
Common SQL Codes	178
Common DL/I Status Codes	180
Common VSAM Status Codes.	181
OPEN request type	181
CLOSE request type	181
GET/PUT/POINT/ERASE/CHECK/ENDREQ request types	182
COBOL Status Key Values	182

Chapter 23. Rational COBOL Runtime Return Codes, Abend Codes, and Exception Codes	185
Return Codes	185
ABEND Codes	185
CICS Environments	185
IMS, IMS BMP, and z/OS Batch Environments	187
Exception Codes	188

Chapter 24. Codes from Other Products for z/OS Systems.	191
Common System Abend Codes for All Environments	191
LE Runtime Messages	192
Common COBOL Abend Codes	193
Common IMS Runtime Messages.	193
Common IMS Runtime Abend Codes	194
Common CICS Runtime Messages	195
Common CICS Abend Codes	195
COBOL Abends under CICS	196

Chapter 18. Diagnosing Problems for Rational COBOL Runtime on z/OS Systems

The chapter contains diagnosis, modification, or tuning information. Use this information to determine the source of the problem you encountered. Some common program definition, database, and system errors that might cause problems are described. This chapter also explains how to obtain error listings and diagnose runtime errors.

Detecting Errors

You can find most logic errors by using the EGL debugger before you generate your program.

During generation, a validation step checks your program for any remaining syntax errors. In addition, validation also checks that your use of language elements is consistent with both the runtime environment and the resource association information you select for each file. For example, the `sysLib.purge()` system function is only valid in a CICS environment.

When you run your generated program, different types of errors are detected by Rational COBOL Runtime, COBOL, the subsystem (IMS or CICS), or z/OS. The error handling varies depending on which product detects the error, the type of error, and the runtime environment.

For diagnostic information of interest at development time, refer to the EGL online help system. For information about how to control the error reporting at runtime, see “Controlling Error Reporting” on page 140. For information about how the various types of errors are reported in the runtime environments, see “Error Reporting Summary” on page 141.

For those errors detected by Rational COBOL Runtime that result in a Run Unit Error, the processing varies based on the runtime environment:

- Error messages for CICS are written to the transient data queue specified through the diagnostic control options. You can print those messages by using the diagnostic printing utility (see “Diagnostic Message Printing Utility” on page 124) or by using CICS utilities (for example, CEBR). For more information, see “Diagnostic Control Options for z/OS CICS Systems” on page 125.
- Error messages for IMS/VS are written to the IMS message queue identified by the `errorDestination` build descriptor option. You can print those messages by using the diagnostic printing utility (see “IMS Diagnostic Message Print Utility” on page 135).

Reporting Errors

Rational COBOL Runtime provides functions that help you determine the cause of a runtime problem. All runtime errors that Rational COBOL Runtime traps are accompanied by error messages and supporting information to help diagnose the problem. Table 22 on page 141 through Table 25 on page 144 show the error diagnostic actions that can be taken based on the severity of the error and the runtime environment.

Controlling Error Reporting

Controlling error reporting requires different actions in CICS, IMS, and z/OS batch environments.

Controlling Error Reporting in CICS

In the CICS environment, error actions are controlled through the online diagnostic controller utility installed as transaction ELAC.

The utility allows you to specify what type of dump is requested, the name of the transient data queue to which Rational COBOL Runtime diagnostic messages are written, the CICS journal number and identifier for error messages, and whether or not a transaction is disabled when a run unit error is detected. The utility lets you reset the default options for all transactions and override the default options for individual transactions.

See “Diagnostic Control Options for z/OS CICS Systems” on page 125 for more details about the diagnostic controller utility.

Controlling Error Reporting in IMS Environments

The following error responses are controlled by build descriptor options for the IMS/VS and IMS BMP environments:

- Write error messages to the error destination message queue. The destination is determined by the **errorDestination** build descriptor option.
- Write error messages to the system log. The log ID is determined by the **imsLogID** build descriptor option. If the **imsLogID** option does not appear in the build descriptor file, error messages will not be written to the system log.
- Put the message that caused the problem for transaction-oriented IMS BMP programs back on the message queue. **restoreCurrentMsgOnError=YES** indicates that the message being processed when the error occurred should be placed back on the message queue before the program ends. **restoreCurrentMsgOnError=NO** indicates that the message being processed should be deleted and not placed back on the message queue. This option is applicable only to a run unit error when Rational COBOL Runtime detects the error. It does not apply to transaction-oriented BMPs that use **VGLib.VGTDLI()**, **dliLib.AIBTDLI()**, or **dliLib.EGLTDLI()** to read the message queue.
- Issue ROLL call or abend for a run unit error. **imsFastPath=NO** results in a ROLL call. **imsFastPath=YES** results in a 1602 abend.

The actions controlled by the runtime JCL are as follows:

- Print message. This is done only if there is an ELAPRINT DD statement in the runtime JCL.
- Snap dump. If the message indicates a snap dump is taken, the snap dump is produced only if there is an ELASNAP DD statement in the runtime JCL.
- Abend 1602 or 1600. This creates a dump only if the runtime JCL contains a SYSUDUMP or SYSABEND DD statement.

Abend code 1602 is the user code issued by Rational COBOL Runtime when it ends the run unit for an **imsFastPath="YES"** program because of an error.

Abend code 1600 is the user code issued by Rational COBOL Runtime in all other situations when it ends program processing because of an unrecoverable error.

IMS takes the following actions, based on the way Rational COBOL Runtime ends the program:

- If a rollback (ROLB) call is issued, the database changes are backed out, the logical unit of work ends, the next message is read from the message queue, and processing continues.
- If a ROLL call is issued, the database changes are backed out, the logical unit of work ends, and IMS stops the program with a user 778 abend. The transaction and PSB are not stopped and can be scheduled again without operator intervention.
- If either a 1600 or a 1602 abend is issued, the database changes are backed out, the logical unit of work ends, and IMS stops the program. The transaction and PSB are also stopped, and they require operator intervention to start them again.

Use ELASNAP so that sufficient data is captured the first time an error occurs.

Controlling Error Reporting in z/OS Batch

The actions controlled by the runtime JCL are as follows:

- Print message. This is done only if there is an ELAPRINT DD statement in the runtime JCL.
- Snap dump. If the message indicates a snap dump is taken, the snap dump is produced only if there is an ELASNAP DD statement in the runtime JCL.
- Abend 1600. This creates a dump only if the runtime JCL contains a SYSUDUMP or SYSABEND DD statement.

Error Reporting Summary

The following tables summarize the error processing actions for Rational COBOL Runtime.

Transaction Error

This error affects only the current CICS task or current IMS/VS transaction. In CICS, the transaction is still available to other end users. In IMS/VS, processing continues with the next message.

Table 22. Error Processing Actions For Rational COBOL Runtime Detected Errors

Environment	Action
CICS	<ul style="list-style-type: none"> • Write error messages to error destination (diagnostic controller option) • Write error messages to CICS journal data set (diagnostic controller option) • CICS dump, dump code ELAD, as determined by message. The type of dump issued for a particular transaction is a diagnostic control option. • Issue a rollback request • Display error messages on terminal, if possible • Set return code to 693 (called programs only)
IMS BMP	See run unit error
IMS/VS (Initial generated program is a main or called basic program)	See run unit error

Table 22. Error Processing Actions For Rational COBOL Runtime Detected Errors (continued)

Environment	Action
IMS/VS (Initial generated program is a main Text UI program)	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option) • Write error messages to system log (imsLogID build descriptor option) • Print messages (ELAPRINT DD statement) • Snap dump determined by the message (ELASNAP DD statement) • Display error messages on current LTERM • Issue a rollback (ROLB) request • Read next message from the queue
z/OS batch	See run unit error

Run Unit Error

The error is likely to occur for every user. In CICS, the transaction might be disabled. In IMS/VS, a new copy of the program is used if there are additional messages on the queue.

Table 23. Error Processing Actions For Rational COBOL Runtime Detected Errors

Environment	Action
CICS	<ul style="list-style-type: none"> • Write error messages to error destination (diagnostic control option), if possible • Write error messages to CICS journal data set (diagnostic control option), if possible • Disable transaction (diagnostic control option) • CICS dump, dump code ELAD, as determined by message. The type of dump issued for a particular transaction is a diagnostic control option. • Issue a rollback request • Display error messages on terminal, if possible • Set return code to 693 (called programs only) • Return
IMS BMP	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option) • Write error messages to system log (imsLogID build descriptor option) • Print messages (ELAPRINT DD statement) • Snap dump determined by the message (ELASNAP DD statement) • Issue a rollback (ROLB) request • Insert message segment or segments into the queue again (restoreCurrentMsgOnError build descriptor option set to YES) • Set return code to 693 • Return
IMS/VS (Initial generated program is a main or called basic program)	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option), if possible • Write error messages to system log (imsLogID build descriptor option), if possible • Print messages (ELAPRINT DD statement), if possible • Snap dump determined by the message (ELASNAP DD statement) • Issue ROLL request if generated with build descriptor imsFastPath="NO" • Abend 1602 if generated with build descriptor imsFastPath="YES"

Table 23. Error Processing Actions For Rational COBOL Runtime Detected Errors (continued)

Environment	Action
IMS/VS (Initial generated program is a main Text UI program)	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option), if possible • Write error messages to system log (imsLogID build descriptor option), if possible • Print messages (ELAPRINT DD statement), if possible • Snap dump determined by the message (ELASNAP DD statement) • Display error messages on current LTERM • Issue ROLL request if generated with build descriptor imsFastPath="NO" • Abend 1602 if generated with build descriptor imsFastPath="YES"
z/OS batch	<ul style="list-style-type: none"> • Print message (ELAPRINT DD statement) • Snap dump determined by the message (ELASNAP DD statement) • Issue a rollback request if DL/I or DB2 databases were used • Set return code to 693 • Return

Catastrophic error

This error indicates storage is corrupted or standard error reporting processing ends abnormally.

Table 24. Error Processing Actions For Rational COBOL Runtime Detected Errors

Environment	Action
CICS	<ul style="list-style-type: none"> • Write error messages to error destination (diagnostic control option), if possible • Write error messages to CICS journal data set (diagnostic control option), if possible • Disable transaction (diagnostic control option) • Display error messages on terminal, if possible • ABEND ELAE. The type of dump issued for a particular transaction is a diagnostic control option.
IMS BMP	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option), if possible • Write error messages to system log (imsLogID build descriptor option), if possible • Print messages (ELAPRINT DD statement), if possible • Issue a rollback (ROLB) request • Abend 1600 (SYSUDUMP or SYSABEND DD statement)
IMS/VS (Initial generated program is a main or called basic program)	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option), if possible • Write error messages to system log (imsLogID build descriptor option), if possible • Print messages (ELAPRINT DD statement), if possible • Abend 1600 (SYSUDUMP or SYSABEND DD statement)

Table 24. Error Processing Actions For Rational COBOL Runtime Detected Errors (continued)

Environment	Action
IMS/VS (Initial generated program is a main Text UI program)	<ul style="list-style-type: none"> • Write error messages to error destination (errorDestination build descriptor option), if possible • Write error messages to system log (imsLogID build descriptor option), if possible • Print messages (ELAPRINT DD statement), if possible • Display error messages on current LTERM, if possible • Abend 1600 (SYSUDUMP or SYSABEND DD statement)
z/OS batch	<ul style="list-style-type: none"> • Print messages (ELAPRINT DD statement), if possible • Abend 1600 (SYSUDUMP or SYSABEND, DD statement)

Rational COBOL Runtime Error

A Rational COBOL Runtime error occurs at a point where the standard error reporting process is not active.

Table 25. Error Processing Actions For Rational COBOL Runtime Detected Errors

Environment	Action
All environments	<ul style="list-style-type: none"> • Abend, ABEND code indicates the reason for the error

See Table 29 on page 151 for information concerning the contents of the registers when either a 1600, 1602, or an ELAE abend occurs.

Using the Rational COBOL Runtime Error Panel

When an error occurs, Rational COBOL Runtime attempts to display error messages on the current terminal. The panels used in displaying error messages are defined as FormGroup ELAxxx where xxx is the language code.

The following figure shows the error panel (form ELAM02 in the FormGroup) as it is shipped with the product. The panel shows the same diagnostic information that is written to the error destination queue, system log or journal, or ELAPRINT file. If there are more error messages than can fit on a single panel, the last line on the panel prompts the user to press a key to display additional error messages.

```

                                Unexpected Program Failure

An unexpected input/output or program error occurred in the
program you were running. Please make a note of the program
name, date, time, and initial error messages and report them to your
system administrator.

Program name ... ART22
Date ..... 08/21/90
Time ..... 13:04:23

Error Messages:

ELA00093I An error occurred in program ART22, function ART229
ELA00131P MSGQ error, file = UNKNOWN, function = CHG, status code = A1
ELA00066I DL/I I/O area = UNKNOWN
                                EDDDED4
                                45256650

```

Figure 32. Panel ELAM02 (example).

Printing Diagnostic Information for IMS

Diagnostic messages are sent either to a print file for batch jobs or to a message queue for IMS BMPs or online transactions. A diagnostic utility is provided to print messages written to a message queue. Optionally, based on the **imsLogID** build descriptor option, the diagnostic information can be written to the IMS log.

errorDestination Message Queue

Table 26 shows the format of the information in the IMS message queue when the **errorDestination** build descriptor option is used. The default queue name is ELADIAG.

Table 26. *errorDestination* IMS Message Queue

Field	Length in Bytes	Type of Data	Description
Record length	2	Binary	The length of the record.
Reserved	2	Binary	A reserved field that must contain binary zeros.
IMS transaction code	8	Character	The name used to identify the IMS message queue that was specified with the errorDestination build descriptor option.
Date	8	Character	Date of the transaction from the I/O PCB (MM/DD/YY).
Time	8	Character	Time of the transaction from the I/O PCB (HH:MM:SS).
NLS	3	Character	The value specified for the targetNLS build descriptor option

Table 26. *errorDestination* IMS Message Queue (continued)

Field	Length in Bytes	Type of Data	Description
Message number	9	Character	The message number: Bytes 1-3 Message File Identifier (ELA) Byte 4 Application Identifier (0) Bytes 5-8 Message Number Byte 9 Message Type. A message type of 'C' indicates this record is a continuation of the specified message from a previous record in the queue.
Message number separator (reserved position)	1	Character	Byte 10 Blank
Message Text	Variable	Character	The text from the message file with specified message inserts.

IMS Log Format

Table 27 shows the format of the information in the IMS log.

Table 27. *IMS Log Record*

Field	Length in Bytes	Type of Data	Description
Record length	2	Binary	The length of the record.
Reserved	2	Binary	A reserved field that must contain binary zeros.
Log ID	1	Character	The value specified with the imsLogID build descriptor option.
Date	8	Character	Date of the transaction from the I/O PCB (MM/DD/YY).
Time	8	Character	Time of the transaction from the I/O PCB (HH:MM:SS).
NLS	3	Character	The value specified for the targetNLS build descriptor option

Table 27. IMS Log Record (continued)

Field	Length in Bytes	Type of Data	Description
Message number	9	Character	The message number: Bytes 1-3 Message File Identifier (ELA) Byte 4 Application Identifier (0) Bytes 5-8 Message Number Byte 9 Message Type. A message type of 'C' indicates this record is a continuation of the specified message from a previous record in the log.
Message number separator (reserved position)	1	Character	Byte 10 Blank
Message Text	Variable	Character	The text from the message file with specified message inserts.

Running the Diagnostic Print Utility

An IMS BMP program is provided to print diagnostic information that is written to the message queue specified by the **errorDestination** build descriptor option. The JCL needed to print the diagnostic information is supplied as member ELAMQJUD of ELA.V6R0M1;ELAJCL.

The message queue identified by the IN parameter is the name of the queue that was specified in the **errorDestination** option when the application was generated. See “IMS Diagnostic Message Print Utility” on page 135 for more information.

Printing Diagnostic Information for CICS

Diagnostic messages are sent to a transient data queue for CICS transactions. A diagnostic print utility is provided to print messages written to a transient data queue. Optionally, as specified by the diagnostic controller utility, the diagnostic information can also be written to an CICS journal data set.

CICS Diagnostic Message Layout

Table 28 shows the format of the information in each error message record written to a transient data queue or CICS journal.

Table 28. Diagnostic Message Layout

Field	Length in Bytes	Type of Data	Description
SYSID name	4	Character	The name of the CICS system that the error message was created on.
TRANID name	4	Character	The name of the CICS transaction code that started the logical unit-of-work.

Table 28. Diagnostic Message Layout (continued)

Field	Length in Bytes	Type of Data	Description
Task identifier	8	Character	The task identifier assigned by CICS to each transaction instance that is processed. This number is reset to 0 when CICS is cold-started. This is taken from EIB field EIBTASKN.
Error destination queue ID	4	Character	The name of the CICS transient data queue. This field is blank if the record is written to the CICS journal.
Date	8	Character	Date of the transaction (MM/DD/YY)
Time	8	Character	Time of the transaction (HH:MM:SS)
NLS	3	Character	The value specified for the targetNLS build descriptor option
Message number	9	Character	The message number: Bytes 1-3 Message File Identifier (ELA) Byte 4 Application Identifier (0) Bytes 5-8 Message Number Byte 9 Message Type. A message type of 'C' indicates this record is a continuation of the specified message from a previous record in the queue.
Message number separator (reserved position)	1	Character	Byte 10 Blank
Message text	110	Character	The text from the message file with specified message inserts

Running the Diagnostic Print Utility

Use the ELAU transaction to print the messages routed to a transient data queue. See “Diagnostic Message Printing Utility” on page 124 for more information about running the CICS diagnostic print utility.

Analyzing Errors Detected while Running a Program

Use the error messages and diagnostic messages to determine the cause of the problem. If the error is detected by another product (for example, COBOL), check the information in Chapter 22, “Common System Error Codes for z/OS Systems” and Chapter 24, “Codes from Other Products for z/OS Systems” and the documentation for the other product.

If you cannot determine the cause of the problem using this information and if the problem can be created again in the test environment, use the EGL debugger to isolate and debug the error before generating the program again.

For debugging in the runtime environment, you can use the runtime diagnostic facility (EDF) for CICS programs or the batch terminal simulator (BTS II) for IMS programs. In addition, if you use the TEST COBOL compile option, you can use the COBOL debugging facilities.

Refer to the CICS, IMS, and COBOL manuals for your versions of these products for additional information on their debugging facilities.

If you get a JCL error for the runtime JCL, check the generation output for the programs involved for any error messages related to JCL generation. In addition, ensure the tailoring that was done on the runtime JCL templates is correct. Also check any changes made to customize the sample runtime JCL.

When abends occur, the problem determination might require assistance from the IBM Support Center. In this case, be prepared to provide IBM with the following information:

- COBOL source file created using the **commentLevel=1** build descriptor option.
- Formatted dump
- Rational COBOL Runtime diagnostic information written to either the error diagnostic queue or listed in the printout for ELAPRINT
- CICS journal or IMS log, as appropriate

IBM requests a COBOL debugger trace listing only if the information is needed for problem determination. IBM will give you the information on how to specify the trace options if the information is necessary.

Chapter 19. Finding Information in Dumps

Information about the problem program can be determined by finding the address of the Rational COBOL Runtime control block in a dump.

Rational COBOL Runtime ABEND Dumps

If the dump code is 1600, 1602, or ELAE, the dump was initiated because Rational COBOL Runtime detected an error. Register 2 at ABEND points to the Rational COBOL Runtime control block. Register 4 points to a linked list of messages formatted as shown in Figure 33.

Table 29. Registers when a SNAP dump is taken or a Rational COBOL Runtime ABEND occurs.

Reg.	Value
2	Points to Rational COBOL Runtime control block. At offset 272 (hexadecimal offset 110) from the start of the Rational COBOL Runtime control block is the address of the initial program profile block, which provides information about the first EGL-generated program that was started. At offset 276 (hexadecimal offset 114) from the start of the Rational COBOL Runtime control block is the address of the current program profile block, which provides information about the EGL-generated program that was running at the time of the abend.
4	Points to the message buffer that contains all messages.

The following diagram shows the format of the message buffer that contains all the messages in the dump.

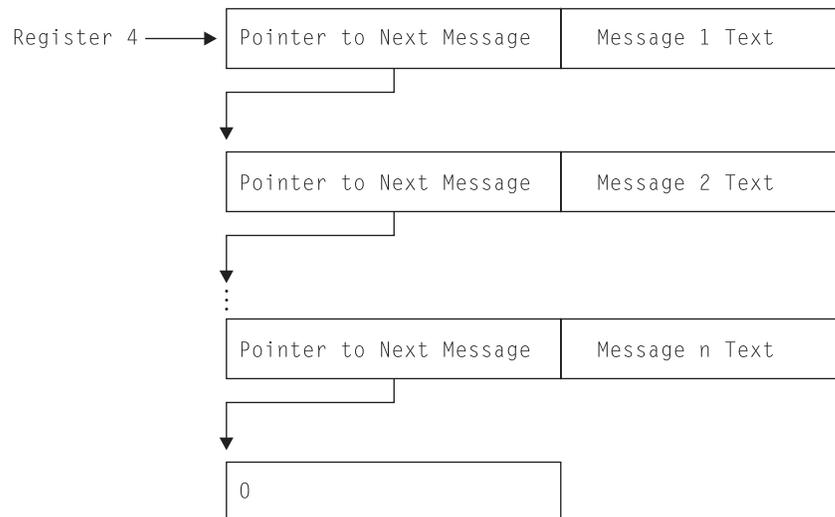


Figure 33. Message Buffer Format

COBOL or Subsystem ABEND Dumps

If the dump is not a Rational COBOL Runtime abend, you can use the following method to locate the Rational COBOL Runtime control block:

- On CICS systems, locate the CICS Task Work Area (TWA) in the dump. Locate the string *EZERTS-CONTROL* in the TWA. This string is the identifier at the start of the Rational COBOL Runtime control block. The * and - characters might be converted to periods in a formatted dump.
- On other systems, locate the string ELARHAPP followed immediately by a program name. ELARHAPP is the identifier at the start of a program profile block. The four-byte address at hex offset 20 in the program profile block is the Rational COBOL Runtime control block address. If 0, the program might not yet be activated. Do a search for another ELARHAPP control block followed by a program name.

For information in the program profile control Block, see Table 31.

Information in the Rational COBOL Runtime Control Block

The following information is in the Rational COBOL Runtime control block:

Table 30. Information in the Rational COBOL Runtime Control Block

Offset in hex	Length in bytes	Contents
0	16	Control block identifier - *EZERTS-CONTROL*
104	4	CICS EIB Pointer
110	4	Program profile address for current program
114	4	Program profile address for initial program
118	8	Terminal identifier
120	8	User identifier
128	8	Transaction identifier
150	12	dliLib.psbData
1CC	18	Current function

Information in a Program, Print Services, or DataTable Profile Block

Each generated COBOL program contains a profile control block in COBOL working storage initialized with information about the program. The first eight bytes contain an eye-catcher constant identifying whether the program was generated from a program, FormGroup or DataTable part. The second eight bytes contain the program name. Other information in the profile block is shown in the following table:

Table 31. Locator Format for Generated COBOL Program Dumps

Offset in hex	Length in hex	Contents
00	08	Program type identifier: ELARHAPP — program, library, service ELAAHMG — print services program ELARMTTP — DataTable program
08	08	Program name
10	08	Program generation date (MM/DD/YY)
18	08	Program generation time (HH:MM:SS)
20	04	Rational COBOL Runtime control block address
24	02	Generator version

Table 31. Locator Format for Generated COBOL Program Dumps (continued)

Offset in hex	Length in hex	Contents
26	02	Generator release
28	02	Generator modification level
2A	10	Reserved
34	08	Target runtime system

How to Find the Current Position in a Program at Time of Error

The Rational COBOL Runtime control block identifies the currently running program and function at the time of the error (Table 30 on page 152). Associated error messages identify the EGL statement number for errors detected by Rational COBOL Runtime that need statement identification to resolve the problem. For performance reasons, the generated COBOL program does not keep track of the EGL statement number for each generated statement. If a program exception occurs in a generated program, you can determine the EGL statement number by finding the COBOL statement that was not successful in a COBOL program listing that contains the EGL statements generated as comments.

Chapter 20. Rational COBOL Runtime Trace Facility

The Rational COBOL Runtime trace facility can be used by the IBM Support Center to aid in problem determination, or by the program user to trace program activity.

There are two levels of tracing available:

- EGL program source-level tracing
- Rational COBOL Runtime runtime level tracing

With source-level tracing, you can request traces of EGL statements, traces of the data, and error codes after every SQL call in a program, except SQL calls made with the **execute** statement. Source-level tracing is enabled with the use of the **sqlIOTrace**, **sqlErrorTrace**, and **statementTrace** build descriptor options. Source-level tracing for the **statementTrace** build descriptor option is automatically activated when you generate with **statementTrace** set to YES and deactivated when you generate with **statementTrace** set to NO. You must activate source-level tracing in the runtime environment for the **sqlIOTrace** and **sqlErrorTrace** build descriptor options by specifying trace filter criteria. See “Activating a Trace” on page 156 for more information on activating traces.

With runtime-level tracing, you can request a data stream trace, a Rational COBOL Runtime internal dump trace, or a service routine trace. Runtime-level tracing does not require the use of a build descriptor option. Runtime-level tracing is activated in runtime environment by specifying trace filter criteria. See “Activating a Trace” on page 156 for more information on activating traces.

Use these functions only with the assistance of the IBM Support Center. If you use these functions without assistance, large amounts of trace output might be produced based on trace option selection.

Enabling EGL Program Source-Level Tracing with Build Descriptor Options

You must specify the **sqlIOTrace**, **sqlErrorTrace**, and **statementTrace** build descriptor options in order to get source-level trace output. EGL generation creates the necessary COBOL code to accomplish the type of tracing that you request.

The trace build descriptor options are **sqlIOTrace**, **sqlErrorTrace**, and **statementTrace**. When using these options, you must specify a value of YES or NO. Each of these build descriptor options tells the COBOL generator whether or not to generate code to allow runtime tracing of a particular aspect of execution - SQL I/O, SQL Errors, and EGL statement execution path.

Note: These options are intended for the use of support personnel and should only be used when a trace is requested as part of a support effort. Normal application debugging should be done through the use of the EGL Debugger.

Activating a Trace

Source-level tracing for the **statementTrace** build descriptor option is automatically activated when you generate with **statementTrace** set to YES and deactivated when you generate with **statementTrace** set to NO.

All other tracing is activated during run time either by using the ELAZ transaction in the CICS or IMS/VS environments, or by specifying the ELATRACE DD name in the runtime JCL for the z/OS batch or IMS BMP environments.

Activating a Trace Session for CICS or IMS/VS

Rational COBOL Runtime supplies a utility to activate tracing in the CICS or IMS/VS environments. To start the utility, enter the utility transaction code, ELAZ. The utility transaction must start prior to running the transaction to be traced.

The ELAZ transaction must run in the same region as the transactions to be traced. In IMS, a message processing region can be altered to handle a unique class and the ELAZ transaction and the transactions to be traced set to this class, in order to ensure that the transactions run in the same region. In CICS, enter the ELAZ transaction and the transaction to be traced from terminals attached to the same CICS region.

Figure 34 shows the initial panel for the ELAZ transaction that enables you to specify which transactions are to be traced. You use a secondary panel to specify filter criteria for a specific transaction that control what information is traced for that transaction.

Note: For IMS/VS, specify the name of the initial program instead of the initial transaction.

```
ELAZ01                Rational COBOL Runtime
                       Trace Transaction Selection

Specify the transaction you want to trace; then press Enter.

To select specific programs and services for tracing, place the cursor
on a transaction name and press F4.

Transaction codes or initial program names

_____  _____  _____  _____
_____  _____  _____  _____
_____  _____  _____  _____
_____  _____  _____  _____

ENTER F1=HELP F3=EXIT F4=FILTER F9=REFRESH F10=STOP TRACE
```

Figure 34. Rational COBOL Runtime Trace Transaction Selection Panel

Rational COBOL Runtime then presents the panel shown in Figure 35 on page 157 for trace filter selection.

```

ELAZ02                Rational COBOL Runtime
                       Trace Filter Selection

Transaction code or Initial Program _____

Fill in the appropriate fields, then press Enter.

3270 Data Stream.....N          APP Statement Trace.....N
Terminal ID.....N                SQL/IO Trace.....N
Trace to File.....N              SQL/ERR Trace.....N
IDUMP Trace.....N

FILENAME ELATOUT  NODE  *      USERID EZEUSRID CLASS A FORM  *

Programs
-----
Services
-----

ENTER  F1=HELP  F3=RETURN  F9=REFRESH

```

Figure 35. Rational COBOL Runtime Trace Filter Selection Panel

The filter criteria include the following:

3270 Data Stream (Y or N)

Specifies whether to trace 3270 data streams

If yes (Y), the 3270 data streams built or received by EGL are traced. The default is no (N). For IMS/VS environments, 3270 Data Stream Trace option is not allowed.

Terminal ID

Specifies a terminal identifier

If specified, only transactions initiated from that terminal are traced. If not specified, service requests from any terminal are traced.

Trace to File (Y or N)

Specifies whether the trace output goes to a file

If yes (Y), the trace output of Rational COBOL Runtime is sent to the ELAT transient data queue in CICS and to an IMS/VS message queue for transaction ELATOUT in IMS/VS. This trace is also written to an in-storage wrap-around trace buffer.

If no (N), the trace output goes to an in-storage wrap-around trace buffer. The size of this trace buffer is defined during customization of Rational COBOL Runtime.

Y must be specified if you specify Y (yes) for the SQL/IO Trace or SQL/ERR Trace options. All trace output for SQL/IO and error tracing is sent to a file, not to the in-storage wrap-around trace buffer.

Note: For IMS/VS, you cannot trace to file if the tracing transaction uses the modifiable express PCB (ELAEXP) because Rational COBOL Runtime uses this PCB to write to the message queue. Unpredictable results can occur.

IDUMP Trace (Y or N)

Specifies whether to dump Rational COBOL Runtime internal storage areas

If yes (Y), the trace facility provides dumps of certain Rational COBOL Runtime internal storage areas. The default is no (N), no internal storage dumps.

APP Statement Trace (Y or N)

This field is ignored for EGL-generated programs; it is only used for programs that are generated by VisualAge Generator Developer. For more information, see the *VisualAge Generator Server Guide for MVS, VSE, and VM*.

SQL/IO Trace (Y or N)

Specifies whether to trace SQL/IO

If yes (Y), the trace facility provides traces of the data and error codes on the return from the SQL call. The EGL function name, the I/O statement, the I/O object, the SQL function name, and the EGL data item name, length, type, and contents are given. Specify the **sqlIOTrace=YES** build descriptor option to enable this type of tracing. The default is no (N).

SQL/ERR Trace (Y or N)

Specifies whether to trace SQL error information

If yes (Y), the trace facility provides traces of the error information that comes back from SQL on every database call. The SQLCODE, SQLERRP, SQLSTATE, SQLWARN, SQLERRD, SQLEXT, and SQLERRMC codes are given. Specify **sqlErrorTrace="YES"** to enable this type of tracing. The default is no (N).

FILENAME

The system resource name for the trace output. The default is ELATOUT.

NODE

1 to 8 characters that specify the system node ID. The default is the current system node ID.

USERID

1 to 8 characters that specify the user ID. The default is the value of the EZEUSRID **sysVar.userID** system variable special function word.

CLASS

A single character that specifies the print class. The default is A.

FORM

1 to 4 characters that specify the form number for print output. The default is your location's standard form.

Programs

Specifies whether to limit the trace to certain programs or print services programs

If specified, only the requested programs are traced.

Services

Specifies whether to limit the trace to certain Rational COBOL Runtime services

If specified, only the requested services are traced. Otherwise all service numbers are traced if the other criteria are met.

Note: The entry to ELARSINI (initialization service) and the exit from ELARSTRM (cleanup service) are not traced. ELARSINI initializes the trace facility. ELARSTRM ends the trace facility.

If you are running a trace to aid in problem determination, enter the filter criteria as directed by the IBM support center.

Activating a Trace Session for z/OS Batch or IMS BMP

Tracing is activated by providing trace filters in a preallocated data set with the DD name ELATRACE before starting the program or job to be traced. ELATRACE contains control statements which control the programs and events to be traced. The attributes for the data set are LRECL=80, DSORG=PS, and RECFM=FB. If the ELATRACE data set is empty or allocated as DD DUMMY, all services are traced, data streams are not traced, and SQL I/O and SQL errors are not traced even if enabled through **sqlIOTrace** or **sqlErrorTrace** build descriptor options. Figure 36 shows the correct syntax for the trace control statements.

```

:FILTER DATASTREAM=Y|N
:FILTER TRACETOFILE=Y|N
:FILTER APPSTMT=Y|N
:FILTER SQLIO=Y|N
:FILTER SQLERR=Y|N
:FILTER IDUMP=Y|N
:APPLS

:
[ name ]

:
: EAPPLS
: SERVICES

:
[ service number ]

:
: ESERVICES
: EFILTER

```

Figure 36. ELATRACE Data Set Entries

Notes:

1. Only one program name or service number can be entered on each line.
2. The :FILTER and :EFILTER tags are required if any other tags are included in the ELATRACE data set.
3. More than one filter can be specified on a line. The filters must be separated by 0 or more blanks. The example below shows sample :FILTER statements that are valid and equivalent:

```

:FILTER DATASTREAM=Y
:FILTER SQLERR=Y

:FILTER DATASTREAM=YSQLERR=Y

:FILTER DATASTREAM=Y SQLERR=Y

:FILTER DATASTREAM=Y           SQLERR=Y

```

The filters cannot be continued on the next line. The statement shown below is not valid:

```

:FILTER DATASTREAM=Y SQLERR=
      Y

```

The control card tags and attributes that control filtering have the following meaning:

- :FILTER** Options controlling what information is traced and where trace data is written
- The following attributes can be used with the :FILTER statement:
- **DATASTREAM=Y|N**
If DATASTREAM=Y is specified, the 3270 data streams built or received by Rational COBOL Runtime are traced. The default value is N, no data stream tracing.
 - **TRACETOFILE=Y|N**
If TRACETOFILE=Y is specified, the trace output is directed to the preallocated data set named ELATOUT in addition to being directed to an in-storage wrap-around trace buffer.
If TRACETOFILE=N is specified, the trace output goes to an in-storage wrap-around trace buffer. The size of this trace buffer is defined during customization of Rational COBOL Runtime. The default for the TRACETOFILE option is N.
TRACETOFILE=Y must be specified if SQLIO=Y or SQLERR=Y are specified. All trace output for SQL I/O and SQL errors is directed to the ELATOUT data set, not to the in-storage wrap-around trace buffer.
 - **APPSTMT=Y|N**
This field is ignored for EGL-generated programs; it is only used for programs that are generated by VisualAge Generator Developer. For more information, see the *VisualAge Generator Server Guide for MVS, VSE, and VM*.
 - **SQLIO=Y|N**
If SQLIO=Y is specified, the trace facility provides traces of the data and error codes on the return from the SQL call. The EGL function name, the I/O statement, the I/O object, the SQL function name, and the EGL data item name, length, type, and contents are given. You must use the **sqlIOTrace="YES"** build descriptor option to enable this type of tracing. The default for the SQLIO option is N.
 - **SQLERR=Y|N**
If SQLERR=Y is specified, the trace facility provides traces of the error information that comes back from SQL on every database call. The SQLCODE, SQLERRP, SQLSTATE, SQLWARN, SQLERRD, SQLEXT, and SQLERRMC codes are given. You must use the **sqlErrorTrace="YES"** build descriptor option to enable this type of tracing. The default for the SQLERR option is N.
 - **IDUMP=Y|N**
If IDUMP=Y is specified, the trace facility provides dumps of certain Rational COBOL Runtime internal storage areas. The default for the IDUMP option is N, no internal storage dumps.
- :APPLS** Program names or print service program names
- If program names are specified, only the specified programs are traced. Otherwise service requests from each generated program are traced. Up to 16 program names can be specified.
- :SERVICES** Service numbers

If service numbers are specified, only those specific services are traced. To trace all service numbers, 999 must be specified. Otherwise, up to 32 service numbers can be specified.

Note: The entry to ELARSINI (initialization service) and the exit from ELARSTRM (cleanup service) are not traced. ELARSINI initializes the trace facility. ELARSTRM ends the trace facility.

Deactivating a Trace Session

To stop all trace activity for CICS or IMS/VS, use the ELAZ transaction to delete the transaction codes from the list of transactions to be traced by using the F10 function key. When a transaction ends and is subsequently restarted, tracing does not start if the transaction code no longer appears in the transaction list.

To stop tracing in z/OS batch or IMS BMP, cancel the program and remove the ELATRACE and ELATOUT DD cards from the runtime JCL.

Printing Trace Output

If the trace output is not directed to a file for the CICS or IMS/VS environments, or the ELATOUT DD statement is not allocated for the z/OS batch or IMS BMP jobs, the trace output is written to a wrap-around trace buffer in memory. The trace output can be seen in dumps taken when programs end abnormally.

Printing the Trace Output in CICS

Trace output for CICS is routed to an extrapartition transient data queue which is directed to a data set named ELATOUT if you direct the trace output to a file by specifying yes (Y) on the ELAZ02 panel. The ELATOUT data set has the attributes of LRECL=133, RECFM=FBA. The file can be printed as directed on the DD statement for ELATOUT in the CICS startup JCL.

Printing the Trace Output in IMS/VS

The trace entries are written to an IMS message queue and can be printed with the ELAEPUTL utility. The sample job stream is shipped as member ELAMQJUD in the installation data set whose low-level qualifier is SELAJCL. You must tailor the sample job stream to set IN=ELATOUT on the EXEC IMSBATCH statement.

Printing the Trace Output in z/OS Batch or IMS BMP

Trace output is directed to the ELATOUT DD statement and is printed as directed on the DD statement. The statement must have the attributes RECFM=FBA,LRECL=133.

Reporting Problems for Rational COBOL Runtime

For instructions on reporting problems, visit the following website:

<http://www.ibm.com/software/awdtools/eglcobol/runtime/support>

Chapter 21. Common Messages during Preparation for z/OS Systems

This chapter contains some error messages from other products. It is not a complete list. For a complete explanation of product messages, refer to the documentation provided with that product.

Common Abend Codes during Preparation

Only the most frequently occurring preparation abend codes are listed in this section. If you receive any other abend code or need a more complete explanation of one of the abend codes, refer to the documentation for that product.

System B37

The temporary work space is filling up. The WSPC parameter that is used in the build scripts to prepare generation output specifies the amount of temporary space allocated.

To resolve the abend, use a symbolic descriptor option named WSPC and set it to a larger value.

System 213, or System 230

Two program developers tried to update the directory of a PDS at the same time. Submit the job again.

This problem can also be prevented by specifying ENQ=YES for the DD statement for the PDS for which the 213 occurred. However, this serializes preparation of servers when their preparation output is placed in the same PDS's.

IMS 3022

The FormGroup that was generated into MFS source resulted in one or more MFS control blocks that exceeds the 32,748-byte limit. The FormGroup cannot be processed by MFS in its current form. Change the FormGroup definition to split the FormGroup into two or more separate FormGroups and then change your program as necessary.

MFS Generation Messages

Only the most frequently occurring MFS generation messages are listed in this section. If you receive other error messages that start with DFS or if you need a more complete explanation of one of the messages, refer to the IMS documentation for your release of IMS.

DFS1141I **name FMT DOES NOT DEFINE
DEVICE INPUT DESCRIPTION FOR
INPUT MESSAGE DESCRIPTION,
FMT NOT PROCESSED**

Explanation: This message can occur when a FormGroup was originally defined and generated with both text and print forms. Then the FormGroup was changed to have only print forms and generated again. This results in a member in the IMS REFERRAL library for the text forms and causes the MFS assemblies to end with errors.

User response: Run the MFSRVC procedure that is supplied with IMS and specify the SCRATCH function to remove this member from the IMS REFERRAL library. Refer to the MFS utilities documentation for your release of IMS for additional information.

DFS1162I **xxxxxxx WARNING: ATTR=nn
SPECIFIED FOR DFLDNAME WHICH
HAD NO EATTR= SPECIFICATION.**

Explanation: You specified the **mfsExtendedAttr="NO"** build descriptor option or

included the **extendedAttributes="NO"** parameter for one or more devices in the **mfsDevice** build descriptor option.

User response: None, provided you wanted to specify devices that do not support extended attributes.

DFS1428I SC=08 LTH=NN,NN EXCEEDS 4 SIGNIFICANT DIGITS. LAST 4 DIGITS USED.

Explanation: This message occurs if a form contains a variable field longer than 8000 bytes for a print form or longer than 1 less than the display size for a text form.

User response: Use form definition to split the

variable field into smaller fields. Change your program to use the smaller fields and then generate the FormGroup and program again.

DFS1587I SC=04 EGCS FIELD SPECIFIED ON AN EVEN COLUMN

Explanation: You defined a DBCS constant or variable field that starts on an even column (in other words, the data starts in an even column).

User response: If the device you are using is an IBM Personal System/55* or is in the IBM 5550 family, you can ignore this message. Otherwise, use form definition to change the definition of the form.

DB2 Precompiler and Bind Messages

Only the most frequently occurring DB2 precompiler and bind messages are listed in this section. If you receive other messages that start with DSN or if you need a more complete explanation of one of the messages, refer to the documentation for your release of DB2.

DSNX039I S PRECOMPILER INTERNAL LIMIT EXCEEDED

Explanation: A limit for the DB2 precompiler has been exceeded. This can occur in programs that contain a large number of SQL I/O functions

User response: Make one or more of the following changes to the program:

- If some of the columns in your SQL tables are defined as NOT NULL, remove the **isSQLNullable=yes** property from the corresponding field in the EGL SQL record definitions. This reduces the number of unique host variables which in turn reduces the number of characters and lines for an SQL statement and the total number of lines for the program. This technique has the biggest impact for the least amount of work and also has the potential of improving performance.
- Review the use of default SQL statements. If the default statements are retrieving more columns than

you actually need, modify the statements to specify only the required columns.

- Shorten the name of the SQL record variable.
- Split the SQL statements into multiple statements. For example, change one **get** statement into multiple **get** statements and retrieve a subset of the columns in each statement.
- Split the program into multiple programs

DSNX100I BIND SQL WARNING

Explanation: One or more DB2 tables have not been created. The tables that do not exist will be identified in an explanation associated with the message by:

```
xxxxxxx IS NOT DEFINED
where xxxxxxx is the table name.
```

User response: Create the necessary DB2 tables and synonyms.

COBOL Compilation Messages

Only the most frequently occurring COBOL compilation messages are listed in this section. If you receive other compilation messages that start with IGY or if you need a more complete explanation of one of the messages, refer to the documentation for your release of COBOL.

IGYPS2015I The paragraph or section prior to paragraph or section EZEMAIN-PROCESS did not contain any statements.

Explanation: These two messages occur if your program has been processed by the DB2 precompiler.

User response: They are normal messages that you can ignore.

IGYPS2023I Paragraphs prior to section EZEMAIN-PROCESS were not contained in a section

IGYOP3091W Code from "?" to "?" can never be executed, and was therefore discarded.

IGYOP3093W The "PERFORM" statement at "?" cannot reach its exit.

IGYOP3094W There may be a loop from the "PERFORM" statement at "?" to itself. "PERFORM" statement optimization was not attempted.

Explanation: These messages occur if your program has been processed using the OPTIMIZE compiler option.

User response: These are normal messages that you can ignore.

IGYPA3013W Data item "?" and "?" had overlapping storage. An overlapping move will occur at execution time.

Explanation: This message occurs if your program attempts to assign the value of a data item to the same data item.

User response: You might want to check that you really intended to do this.

IGYPG3113W Truncation of high-order digit positions may occur due to precision of intermediate results exceeding 30.

Explanation: This message might occur if your program was generated with the **math="COBOL"** build descriptor option.

User response: You might want to change the arithmetic expression identified in the message. For example, you could split the expression into several smaller ones.

If you do not change the expression, ensure that the intermediate values will fall within the precision that COBOL supports. Refer to the programming guide for your release of COBOL for more information about the precision of intermediate results.

IGYSC2025W "EZPCB-?" or one of its subordinates was referenced, but "EZPCB-?" was a "LINKAGE SECTION" item that did not have addressability. This reference will not be resolved successfully at execution.

Explanation: This warning message occurs when PCBs or any data structure is generated in the linkage section, but is not used in a statement.

User response: Ignore the messages and the program will work correctly.

Chapter 22. Common System Error Codes for z/OS Systems

The information within this chapter is diagnosis, modification, or tuning information.

Rational COBOL Runtime messages might include error codes from databases or operating systems that are being used. This could include DB2, DL/I, z/OS VSAM, or CICS EXEC Interface Block (EIB) codes.

This chapter contains only the most common errors that occur during file input and output operations.

The error codes included in this chapter are for the following databases and operating systems:

- CICS
- DB2
- DL/I
- VSAM
- COBOL

Common Error Codes

If you set **v60ExceptionCompatibility** program property to YES, the system variable **sysVar.errorCode** contains an error code indicating a reason that a file I/O statement or a system function invocation is not successful. Codes specific to the system or the access method are returned when the **sysCodes** build descriptor option is set to YES.

If you set the **sysCodes** build descriptor option to NO, the system error codes are converted to EGL error codes. This allows applications developed previously under Cross System Product or VisualAge Generator to receive the same error codes as before.

System Error Code Formats for **sysVar.errorCode**

The following table shows the formats of **sysVar.errorCode** by specific environment:

System	Compatibility Considerations
CICS	<p data-bbox="639 222 1425 394">If sysVar.errorCode is in the form RSnnnnnn, look under nnnnnn in “Common System Error Codes in sysVar.errorCode” on page 170. Otherwise, the first 2 characters of sysVar.errorCode contain the first byte of the EIBFN from the CICS EXEC interface block (EIB). The remaining 6 characters contain bytes 0-2 of the EIBRCODE, also from the CICS EXEC interface block.</p> <p data-bbox="639 424 1425 508">If all of the following are true, then the first 2 characters of sysVar.errorCode contain the first byte of the EIBFN and the remaining 6 characters contain bytes 0-2 of the EIBRCODE:</p> <ul data-bbox="639 520 1425 617" style="list-style-type: none"> <li data-bbox="639 520 1425 546">• The program is running in VisualAge Generator compatibility mode <li data-bbox="639 558 1425 583">• VGVar.handleSysLibErrors is set to 1 <li data-bbox="639 596 1425 617">• A call statement is implemented with CICS LINK <p data-bbox="639 646 1425 693">Refer to the CICS application programmers' guide for an explanation of the EIB codes.</p>

System	Compatibility Considerations
z/OS batch	<p>If sysVar.errorCode is in the form RSnnnnnn, look under nnnnnn in “Common System Error Codes in sysVar.errorCode” on page 170.</p> <p>GSAM: sysVar.errorCode contains the DL/I status code after an I/O statement. The last 6 characters of sysVar.errorCode are blanks.</p> <p>SEQ: sysVar.errorCode contains the COBOL status key value or values in the first 2 characters. The remaining 6 characters are zeros.</p> <p>SEQRS: The contents of sysVar.errorCode depend on the operation that was not successful:</p> <ul style="list-style-type: none"> • If a dynamic allocation is not successful, the first 3 bytes of sysVar.errorCode contain the value S99 (for SVC 99, dynamic allocation), byte 4 is the SVC 99 return code in hexadecimal, and bytes 5-8 contain the error reason code in hexadecimal. • If an OPEN is not successful, sysVar.errorCode contains return code 8 ('00000008'). • If a READ end-of-file condition occurs, sysVar.errorCode contains return code 4 ('00000004'). • If a READ, WRITE, or CLOSE is not successful, sysVar.errorCode contains return code 12 ('00000012'). <p>VSAM: sysVar.errorCode contains the COBOL status key value or values in the first 2 characters followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code).</p> <p>VSAMRS: The operation that is not successful determines the contents of sysVar.errorCode:</p> <ul style="list-style-type: none"> • If a dynamic allocation is not successful the first 3 bytes of sysVar.errorCode contain the value S99 (for SVC 99, dynamic allocation), byte 4 is the SVC 99 return code in hexadecimal, and bytes 5-8 contain the error reason code in hexadecimal. • If an OPEN or CLOSE is not successful, the first 2 bytes of sysVar.errorCode contain the error code from the VSAM application control block (ACB) in hexadecimal. The remaining 6 characters are zeros. • If an operation other than OPEN or CLOSE is not successful, the first 2 characters are zeros followed by 2 characters for the COBOL VSAM return code (VSAM feedback code), 1 character for the COBOL VSAM function code (VSAM component code), and 3 characters for the COBOL VSAM feedback code (VSAM reason code). <p>For VSAM codes, refer to <i>z/OS V1R7 DFSMS Macro Instructions for Data Sets (SC26-7408)</i>. For SVC 99 codes, refer to <i>z/OS V1R7.0 MVS System Codes (SA22-7626)</i>.</p>
IMS/VS	<p>The only files that can be used in this environment are serial files associated with IMS message queues. sysVar.errorCode contains the DL/I status code after an I/O statement to one of these files. The last 6 characters of sysVar.errorCode are blanks.</p>
IMS BMP	<p>IMS message queue: sysVar.errorCode contains the DL/I status code after an I/O statement. The last 6 characters of sysVar.errorCode are blanks.</p> <p>Otherwise, same as z/OS batch in this table.</p>

Common System Error Codes in sysVar.errorCode

The following table gives an explanation of the most common values that you receive in **sysVar.errorCode** when the **sysCodes** build descriptor option is set to YES. If your error code is not listed here, or you would like more information, refer to the table in “System Error Code Formats for sysVar.errorCode” on page 167 and then the appropriate manuals for your environment.

Table 32. **sysVar.errorCode** error codes.

System	Return code	Meaning
z/OS batch	A0000000	VSAM open error - empty indexed file
z/OS batch	BC000000	VSAM open error - file is not in VSAM format
z/OS batch	S9940210	File not available
z/OS batch	S9940440	File not found
z/OS batch	S99417**	File not found
z/OS batch	00000004 on non-VSAM file	End of file
z/OS batch	00000008 on non-VSAM file	Error opening file
z/OS batch	00000012 on non-VSAM file	Error on I/O or closing a file
z/OS batch	0008*004 for nonrelative	End of file
z/OS batch	0008*004 for relative	No record found
z/OS batch	0008*008 for an add statement	Duplicate record
z/OS batch	0008*016 if get next for an indexed record	End of file
z/OS batch	0008*016 if not using get next for an indexed record	No record found
z/OS batch	0008*028	File full
z/OS batch	0008*116	No record found
z/OS batch	*****74	No record found
Note: * represents any character.		
Note: z/OS batch in this table includes IMS BMP		
CICS	ffrrrrr Remote call or vgLib.startTransaction()	Other CICS errors: <ul style="list-style-type: none"> • ff = Hexadecimal representation of EIBFN byte 0 • rrrrrr = Hexadecimal representation of EIBRCODE bytes 0-2
CICS	0A010000 get next for a temporary storage queue	End of file
CICS	0A010000 on direct I/O to a temporary storage queue	No record found
CICS	0A080000 on temporary storage queue	File is full
CICS	060F0000 on VSAM file	End of file
CICS	00000000 Remote call or vgLib.startTransaction()	Successful
CICS	00000203 Remote vgLib.startTransaction()	Transaction identifier that is not valid
CICS	00000204 Remote call	Program name that is not valid
CICS	00000207 Remote call or vgLib.startTransaction()	System identifier that is not valid
CICS	00000208 Remote call	Link out of service or is not valid

Table 32. **sysVar.errorCode** error codes. (continued)

System	Return code	Meaning
CICS	06810000 on VSAM file	No record found
CICS	06820000 on VSAM file	Duplicate record
CICS	06830000 on VSAM file	File is full
CICS	08E10000 on transient data	Format error
CICS	08010000 on a transient data queue	End of file
CICS	08020000 on a transient data queue	File not found
CICS	08080000 on transient data	Transient data queue not open
CICS	12320000	Queue is already in use

EGL Error Codes

The error codes list is sequenced by error code, with the alphabetic error codes (A to Z) occurring before the numeric error codes (0 to 9). If you set the **sysCodes** build descriptor option to NO, **sysVar.errorCode** will contain error codes that are compatible with the Cross System Product codes.

Table 33. Rational COBOL Runtime Error Codes

Error code	Component	Probable Cause
<i>Cnn</i>	File control/request	These error codes do not have an EGL equivalent I/O error value. Either CICS returned an IOERR error or VSAM returned a return code of 12 on file input/output. The <i>nn</i> is replaced by the VSAM reason code from the feedback field. For more information, refer to the <i>z/OS V1R7 DFSMS Macro Instructions for Data Sets (SC26-7408)</i> manual.
<i>Fnn</i>	File control/request	These error codes are CICS EIBRCODES, other than ILLOGIC, IOERR, and those that have EGL equivalent I/O error values. The <i>nn</i> is replaced by the EIBRCODE (byte 0). For more information, refer to the application programming reference for your release of CICS. Note: All error codes, other than the ones that have EGL equivalent I/O error values, cause the program to end. An error message is issued to inform you that the program ended because of a send/receive error. The error message includes the error code.
FE1	File Control/request	Transient data queue - Queue length and EGL record length do not match. The invalidFormat EGL I/O error value is set.
F02	File Control/request	Transient data queue - File not found. The fileNotFound EGL I/O error value is set.
F08	File control/request	An attempt was made to gain access to an extrapartition transient data queue, but the queue has not been opened yet. Exit and use CEMT to open the queue.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
52	Terminal support	You attempted to run an EGL program from an unsupported device (such as a 3278-52 terminal). This device is not supported by EGL.
101	Message processing	The message was truncated.
102	Contents control	The module specified on a LOAD macro is already in storage.
	File control/request	The end of file was reached. The endOfFile EGL I/O error value is set. Note: endOfFile is set when a get next is performed on an empty file.
	Service request	An ITEMERR condition was received from CICS because the maximum number of records allowed in a temporary storage queue (32767) was exceeded.
103	File control/request	You performed an operation on a record that has a duplicate key, or a key in the record for an alternate index is duplicated. The I/O operation completed, and the duplicate EGL I/O error value is set.
104	File control/request	The end of file was reached. The endOfFile EGL I/O error value is set.
115	Service request	An EXEC CICS ENQ was not successful.
116	Service request	An EXEC CICS DEQ was not successful.
20B	Storage allocation	Operands that are not valid were specified on either a GETMAIN or FREEMAIN macro.
20C	Defined data set	The data set name specified on an issued DEFDS command already exists in the external work file.
	Storage allocation	An error occurred while processing a FREEMAIN macro.
200	Service request	An service request was issued that is not valid. This is a system error. Contact the IBM Support Center.
201	File open/connect	The connection already exists. The possible cause is a file with the same name is already in use. The fileNotAvailable EGL I/O error value is set.
	Message processing	Variables were passed to be built into the message, but the message contained no variable fields; or, the message contained variable fields, and no variables were passed for them.
201- 206	Service request	Service request errors occurred while processing a DEFDS command. This is a system error. Contact the IBM Support Center.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
202	File control/request	Record not found. The noRecordFound EGL I/O error value is set.
	Storage allocation	The ORIGIN specified on a FREEMAIN macro does not match storage already in use.
203	File control/request	The record was not found. The EGL I/O error value noRecordFound is set.
	Storage allocation	Either the ORIGIN specified on a FREEMAIN macro does not begin on a doubleword boundary, or 0 LENGTH was specified on a GETMAIN.
204	Storage allocation	An attempt has been made to free storage that has not been allocated or that has already been freed.
205	File control/request	The record was not found. The noRecordFound EGL I/O error value is set. Note: The noRecordFound EGL I/O error value is set when a get next or get previous is preceded by a set record position on an empty indexed file.
	Storage allocation	The LENGTH specified on a FREEMAIN macro is 0.
206	File control/request	You attempted to store a record with a duplicate key while using an index that does not allow duplicate keys. The duplicate EGL I/O error value is set.
207	File control/request	The record was not found. The noRecordFound EGL I/O error value is set.
208	File control/request	An error occurred when you attempted to connect or write to the log file on CICS. A possible reason for the error is that no TDQUEUE entry was found for the log file.
	Service request	The NDSNAME in an ALTDS request is not valid. This is a system error. Contact the IBM Support Center.
	Storage allocation	The storage specified on a FREEMAIN macro is already free.
209	Service request	The name specified by the NDSNAME on an ALTDS command already exists in the external work file. This is a system error. Contact the IBM Support Center.
210- 211	Service request	Miscellaneous errors occurred on an ALTDS request. This is a system error. Contact the IBM Support Center.
212	Service request	An I/O error occurred while copying data from the work file to the external work file during an ALTDS service request.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
213	Service request	The COPIES operand on a SUBMIT.PRINT service request is not valid. This is a system error. Contact the IBM Support Center.
214	Service request	The data set on a SUBMIT.PRINT service request cannot be found. This is a system error. Contact the IBM Support Center.
215	File control/request	You attempted to store a record with a duplicate key while using an index that does not allow duplicate keys. The duplicate EGL I/O error value is set.
216	File open/connect	A connection was attempted to an ESDS file or transient data queue in direct mode. The invalidFormat EGL I/O error value is set.
	Service request	The data set specified on a DEFDS request matches a CONNECT already in use. This is a system error. Contact the IBM Support Center.
217	File open/connect	An attempt was made to subconnect to a serial file. Check to see if a called program is attempting to reference the same serial file that has been referenced by the calling program.
	Service request	A PRINT error has occurred for one of the following reasons: <ul style="list-style-type: none"> An error occurred while writing to the transient data queue on CICS. The most common errors are QIDERR, IOERR, LENGERR, and NOSPACE. An error occurred while writing to the EZEPRINT data definition name (DD name) in z/OS batch or IMS BMP. A possible cause is that the printer file (for example, EZEPRINT) has been allocated incorrectly or has not been allocated at all. For example, the data set allocated for the print output has the wrong record format (anything other than VBA) or the wrong record length (shorter than the print output line length).
218	Service request	The file is not available. The fileNotAvailable EGL I/O error value is set.
22A	File control/request	The available storage space has been exhausted. Try the operation again. If the problem persists, contact your system programmer.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
220	File open/connect	A format error occurred. Either the characteristics of a file are not supported by EGL, or they are incompatible with the EGL record definition. The invalidFormat EGL I/O error value is set. For example, a serial file is trying to access a member of a PDS data set, but the JCL for z/OS batch or IMS BMP does not specify a member name
	File control/request	The record length for a file is larger than the maximum record length defined in the system.
221	Service request	An ENQ was not successful while writing to the transient data queue on CICS. This is a system error. Contact the IBM Support Center.
223	Service request	The attach of the print subtask was not successful, or the print subtask abended. This is a system error. Contact the IBM Support Center.
225	Service request	The print subtask abended. This is a system error. Contact the IBM Support Center.
226	File control/request	An IO error occurred while reading or writing from temporary storage on CICS. This is a system error. Contact the IBM Support Center.
25A	File control/request	The data set cannot be extended because VSAM cannot allocate additional direct-access storage space. Either not enough space is left to make the secondary allocation request, or you attempted to increase the size of a data set while processing with SHROPT=4 and DISP=SHR. The full EGL I/O error value is set.
251	File open/connect	For CICS environments, the file control table (FCT) entry was not found, indicating a real file or transient data queue was not properly defined or generated. For z/OS batch or IMS BMP environments, either the DD name has not been allocated, or the data set for the dynamic allocation does not exist.
280	File control/request	The data set that you are trying to connect to is already in use. A probable cause is that your program has a data set associated with one record variable and you are trying to use another record variable with the same data set. You need to issue a CLOSE on the first record variable to free the data set before trying to use it with another record variable.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
291- 294	Service request	A mapping error occurred.
	Terminal support	A mapping error occurred.
380	File control/request	A deadlock occurred. One transaction is attempting to update a record that is currently locked by another transaction. The deadlock EGL I/O error value is set.
381	File control/request	The control interval for a record is already held in exclusive control by another requester. The deadlock EGL I/O error value is set. For CICS, the returned code is INVREQ. This is assumed to have occurred due to one transaction's attempt to do two get forUpdate statements to the same file. If this is not the case, see the description of INVREQ in the application programming reference for your release of CICS.
389	File control/request	The resource control block could not be found to process the request against. This is a system error. Contact the IBM Support Center.
399	File control/request	You attempted to store a record to a temporary storage queue with a key that exceeds 32767. The key is too large for temporary storage queues, which cannot have more than 32767 records.
4nn	File open/connect	For z/OS batch or IMS BMP environments, the VSAM GENCB for an ACB was not successful. The value of <i>nn</i> is determined from VSAM return codes. If register 15 contains 4, <i>nn</i> is replaced by the contents of register 0. If register 15 does not contain 4 (or 0), <i>nn</i> is replaced by 50 plus the contents of register 15.
5nn	File open/connect	For z/OS batch or IMS BMP environments, an OPEN request is not successful. For VSAM files in z/OS batch or IMS BMP environments, a SHOWCB for the ERROR field is done after the problem with the OPEN request. The value of the ERROR field replaces <i>nn</i> . For non-VSAM sequential files in z/OS batch or IMS BMP environments (QSAM), <i>nn</i> is replaced with a value of 0. For spool files in a CICS environment, <i>nn</i> is also replaced with 0.
5A0	File open/connect	An attempt was made to open a VSAM data set for input, but the data set was empty.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
6nn	File open/connect	The VSAM GENCB for an RPL was not successful. For z/OS batch and IMS BMP environments, the value of <i>nn</i> is determined from VSAM return codes. If register 15 contains 4, <i>nn</i> is replaced by the contents of register 0. If register 15 does not contain 4 (or 0), <i>nn</i> is replaced by 50 plus the contents of register 15.
701	File open/connect	On CICS Version 2.1 or later, the file cannot be opened or connected. The error is not defined in the FCT flags.
702	File open/connect	The VSAM SHOWCB or MODCB macro was not successful. This usually means that the file is not open.
703	File open/connect	The VSAM TESTCB macro was not successful.
705	File open/connect	For CICS only, a connection was attempted to transient data or a temporary storage queue, but a VSAM file has the same name.
706	File open/connect	On CICS Version 2.1 or later, the file is UNENABLED and cannot be opened or connected.
707	File open/connect	On CICS Version 2.1 or later, the file is DISABLED or DISABLING and cannot be opened or connected.
708	File open/connect	On CICS Version 2.1 or later, the user is not authorized to have access to the file.
709	File open/connect	On CICS Version 2.1 or later, an I/O error occurred on the SET data set Open command.
768	File open/connect	The OPEN or connection was not successful due to a GETMAIN error when requesting storage for control blocks associated with sequential files.
8nn	File control/request	These return codes do not have an EGL equivalent I/O error value. Either CICS returned an ILLOGIC error, or VSAM returned a return code of 8 on file input/output. The <i>nn</i> is replaced with the VSAM error code. For more information, see the <i>z/OS V1R7 DFSMS Macro Instructions for Data Sets (SC26-7408)</i> manual.
	File open/connect	A storage allocation was not successful.
80C	Storage allocation	There is insufficient storage to satisfy a GETMAIN request.
802	File open/connect	The resource is not associated.
	Storage allocation	There is insufficient storage for allocation.
803	Contents control	The module specified on a LOAD macro could not be located.

Table 33. Rational COBOL Runtime Error Codes (continued)

Error code	Component	Probable Cause
804	File control/request	This return code is received from CICS and indicates that a QIDERR or ITEMERR occurred while trying to gain access to a temporary storage queue.
805	Contents control	The module specified on a LOAD macro is damaged.
	Message processing	An unsupported option was specified on an INFORM macro.
806	Contents control	For z/OS batch or IMS BMP, the module specified on a LOAD macro could not be located.
807	Contents control	Insufficient storage is available to load the specified module.
81C	File control/request	A temporary storage queue is full. The full EGL I/O error value is set.
987	File control/request	For z/OS batch and IMS BMP, a branch was made to the SYNAD routine as the result of a GET to a non-VSAM serial file. A possible reason is that the file is empty or the blocking factor is wrong.
988	File control/request	For z/OS batch and IMS BMP, a branch was made to the SYNAD routine as the result of a PUT to a non-VSAM serial file. A possible reason is that the file is empty or the blocking factor is wrong. For CICS, a WRITE request to a spool file was not successful.
989	File control/request	An error occurred while trying to close a file.
999	File control/request	An unsupported request was made for a serial file. A probable cause is that the EGL record associated with this file was not defined as a serial record.

Common SQL Codes

After an SQL I/O statement, the SQL code is stored in the `sysVar.sqlData.sqlCode` system variable. Only the most frequently occurring SQL codes are listed in this section. If you receive other SQL codes or if you need a more complete explanation of one of the SQL codes, refer to the documentation for your release of DB2.

- 100** No rows were found by SQL that meet the search criteria specified in the WHERE clause of the SQL statement, or if processing a **get** statement with a position option in conjunction with an **open** or **open forUpdate** statement, the end of the selected rows has been reached. The possible causes are the following:
- The key value(s) were not moved correctly to the host variable(s) used in the WHERE clause.
 - No rows meet the search criteria specified in the WHERE clause.
 - Rational COBOL Runtime stripped trailing blanks for the character host variables used in a LIKE predicate in the WHERE clause. You can set the

sqlIOTrace build descriptor option to YES to enable tracing of the data sent to SQL and the data coming back from SQL. For more information, see Chapter 20, “Rational COBOL Runtime Trace Facility,” on page 155.

-301, -302, -303, -304

The EGL data item definition does not match the definition of the same column in the DB2 table. This can be caused by defining a column as variable length, but not defining the data item in EGL with a variable-length SQL code. This can also be caused by specifying a different length to EGL than what you defined in the DB2 table.

Make the necessary changes in the EGL data item definition to match the DB2 table and generate the program again.

-302 For the IMS/VS environment, you might have allocated the DB2 work database with a 4KB page size instead of the required 32KB page size. Refer to the Rational COBOL Runtime program directory for information about installing a DB2 work database.

-805 The DBRM for the current program was not bound as part of the current DB2 plan. Possible causes are:

- The BIND process was never run for the program.
- An incorrect plan name was specified at startup.
- The plan name specified in the DB2TRAN or DB2ENTRY definition for CICS did not match the plan name used in the BIND process.
- All programs that run together under a single transaction or job step must be bound into the same DB2 plan.

Look at the message inserts to see what DB2 returned as the program name and plan name. If these are what you expect, review the steps used for preparing the program.

-818 The DB2 precompiler-generated time stamp in the load module is different from the database request module (DBRM) used on the most recent BIND for the PLAN being used. The load module and the DBRM from the precompiler must match and one of them is not from the most recently-run precompile. This typically happens when the precompile, link-edit, and bind process is run more than once and either the DBRM library or the load library used for the load module is changed. This creates the opportunity to pick up the old load module at run time if the old load library is first in the search sequence at run time. Alternatively, the BIND process might be using an old DBRM library that contains an old copy of that member.

Ensure that you are running with the most recent copy of the load module and that you are using the same DBRM library on the precompile and BIND steps. On CICS ensure that the latest copy of the load module has been picked up by issuing an CICS NEWCOPY command or by using the Rational COBOL Runtime new copy utility. On IMS/VS ensure that the latest copy of the load module has been picked up by recycling the message region.

-911,-913

A deadlock condition occurred. Possible causes are:

- The isolation level was set for repeatable read.
- There were long periods of time between commit points.
- In EGL, the program issued a **get forUpdate** statement, but failed to issue a related **replace** statement. In VisualAge Generator, the program issued an UPDATE without a REPLACE.

Note: The program should be coded to handle these conditions.

- 922 Connection authorization was not successful. The type of error is indicated in the SQL error message. Some typical causes are not granting authority for the DB2 plan or not creating a synonym for one or more of the DB2 tables.

Make the necessary changes to provide authorization to the DB2 plan and then run the program again. You might also want to refer to the documentation for your release of DB2 for additional causes of the authorization error.

Common DL/I Status Codes

After a DL/I I/O statement, the DL/I status code is stored in the **dliVar.statusCode** system variable. Only the most frequently occurring DL/I status codes are listed in this section. If you receive other DL/I status codes or if you need a more complete explanation of one of the DL/I status codes, refer to the *IMS Messages and Codes Volume 1* manual for your release of IMS.

- AD** The function parameter on the call is not valid. If the function code is correct, the status code can be from an I/O or alternate PCB for a database call. You might have a mismatch between the EGL PSB record definition and the IMS PSB definition.
- AI** A data management open error occurred. Either no DD statements were supplied for logically related databases, or the DD name is not the same as the name specified on the DATASET statement of the DBD. The segment name area in the DB PCB has the DD name of the data set that could not be opened.
- AJ** The format of one of your SSAs is not valid. Either the SSA contains a command code for the call that is not valid, or the SSA uses an R, S, W, or M command code for a segment for which there are no subset pointers defined in the DBD.
- AK** An SSA contains either a field name that is not valid or a name that is not defined in the DBD, or the EGL **dliFieldName** property for the field in the DL/I segment record does not match the name defined to DL/I.
- AM** The call function is not compatible with the processing option in the PCB, the segment sensitivity, the transaction-code definition, or the program type.
- GA** In trying to satisfy an unqualified GN or GNP call, DL/I crossed a hierarchic boundary into a higher level.
- GB** In trying to satisfy a GN, DL/I reached the end of the database.
- GD** The program issued an ISRT that was not qualified for all levels above the level of the segment being inserted. The segment might have been deleted by a DLET using a different DB PCB.
- GE** DL/I is unable to find a segment that satisfies the segment described in a get call.
- GK** DL/I has returned a different segment type at the same hierarchic level for an unqualified GN or GNP.
- GP** The program issued a GNP when parentage is not established, or the segment level specified in the GNP is not lower than the level of the established parent.

- II The program issued an ISRT that tried to insert a segment that already exists in the database.

Common VSAM Status Codes

Only the most frequently occurring VSAM codes are listed in this section. If you receive other VSAM codes or if you need a more complete explanation of one of these values, refer to the *z/OS V1R7 DFSMS Macro Instructions for Data Sets (SC26-7408)* manual.

OPEN request type

Code	Meaning
------	---------

- | | |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 64 | Warning message: OPEN encountered an empty alternate index that is part of an upgrade set. |
| 74 | This is a warning message indicating the data set was not properly closed. Either the implicit verify for the OPEN was unsuccessful, or the user specified that the implicit verify should not be attempted for the OPEN. A previous VSAM program might have ended abnormally. The VERIFY command of Access Method Services can be used to properly close the data set. |
| 80 | The DD statement for this access method control block is either missing or not valid. |
| 94 | Either no record for the data set to be opened was found in the available catalog or catalogs, or an unidentified error occurred while VSAM was searching the catalog. |
| 98 | Security verification was not successful; the password specified in the access method control block for a specified level of access does not match the password in the catalog for that level of access. |
| A0 | The operands specified in the ACB or GENCB macro are inconsistent either with each other or with the information in the catalog record. You might have attempted to open an empty data set for input only (get next statement). |
| A8 | The data set was not available for the type of processing you specified, or an attempt was made to open a reusable data set with the reset option while another user had the data set open. |
| BC | The data set indicated by the access method control block is not a valid type of data set for specification by an access method control block. You might have used a sequential data set as the physical file, but specified VSAM or VSAMRS as the file type for resource association when you generated the program. |
| C0 | An unusable data set was opened for output. |
| C4 | Access to data was requested using an empty path. |

CLOSE request type

Code	Meaning
------	---------

- | | |
|----|--------------------------------------------------------------------------------------------------------------------|
| 04 | The data set indicated by the access method control block is already closed. |
| 88 | Not enough virtual storage was available in the address space of your program for the work area required by CLOSE. |

94 An unidentified error occurred while VSAM was searching the catalog.

GET/PUT/POINT/ERASE/CHECK/ENDREQ request types

Note: The following occur when register 15=8(8).

Code	Meaning
08	An attempt is made to store a record with a duplicate key, or there is a duplicate record for an alternate index with the unique key option.
6C	The RECLLEN specified was one of the following: <ul style="list-style-type: none">• Larger than the maximum allowed• Equal to 0• Smaller than the sum of the length and the displacement of the key field• Not equal to the record(slot) size specified for a relative record data set
70	The KEYLEN specified was too large or equal to 0.
C0	A relative record number that is not valid was encountered.

COBOL Status Key Values

This shows the most frequently occurring COBOL status key values. If you receive other status key values or if you need a more complete explanation for one of these values, refer to the application programming language reference for your release of COBOL.

Status Key	Explanation
10	The end of a file was reached.
22	An attempt was made to write a record with a key that duplicated one that was already in the file.
23	Record not found. This can also be caused by an optional file not being allocated.
35	No DD statement was included in the JCL. This can occur if the program calls another program or transfers to another program using a transfer to program statement, but the DD statements for the second program have not been added to the sample runtime JCL for the main program.
39	The physical file that you specified during resource association does not match the file characteristics that you specified during record definition. The file characteristics include file organization (sequential, relative or indexed), the prime record key, the alternate record keys, and the maximum record size.
44	A variable-length record was written that is not valid. This can occur if the value in the numElementsItem field for the record is larger than the maximum value, or the value in the lengthItem field for the record is larger than the maximum length of the record.
96	No DD statement was included in the JCL for a VSAM file. This can occur if the program calls another program or transfers to another program using a transfer to program statement, but the

DD statements for the second program have not been added to the sample runtime JCL for the main program.

Chapter 23. Rational COBOL Runtime Return Codes, Abend Codes, and Exception Codes

The information within this chapter is diagnosis, modification, or tuning information.

Only the most frequently occurring abend codes are listed in this section. If you receive other abend codes or if you need a more complete explanation of one of the codes, refer to the z/OS messages and codes manual for your release of z/OS.

Return Codes

This section contains a listing of codes set by Rational COBOL Runtime and returned in the COBOL return code of a program.

If a generated program completes normally, the COBOL return code is set to the value in the `sysVar.returnCode`. This code must be less than or equal to 512. Return codes greater than 512 are reserved for Rational COBOL Runtime. The return codes set by Rational COBOL Runtime are:

- 693** The program ended due to an error detected by Rational COBOL Runtime. The error description is reported as described in Chapter 18, "Diagnosing Problems for Rational COBOL Runtime on z/OS Systems."
- 4093** A program generated using EGL ended due to an error detected by Rational COBOL Runtime.

If LE detects an error and returns to the operating system, the LE return code modifier (2000 - error, 3000 - severe error, or 4000 - critical error) is added to the user or Rational COBOL Runtime return code.

ABEND Codes

Rational COBOL Runtime reports errors by error messages whenever possible. Abend codes are issued only in situations where initialization has not progressed to the point where messages can be issued or when the error messages cannot be written to their normal destination.

CICS Environments

For CICS, you can control whether or not a core dump is taken by using the diagnostic controller utility. If a core dump is taken, the dump code is ELAD. See "Controlling Error Reporting in CICS" on page 140 for information on the diagnostic controller utility.

- ELA1** This abend code should never be received. However, if register 1 in a dump contains "ELA1", then a database manager or subsystem interface module, such as ASMTDLI for DL/I access, was not linked with a Rational COBOL Runtime program at product installation. Registers 3 and 4 in the dump usually contain the name of the stub program. The load module where the abend occurred is the module that was not linked correctly.

Refer to the *Program Directory for Rational COBOL Runtime for zSeries* for information on correctly linking the abending load module.

- ELA2** The Task Work Area (TWA) does not exist or is not long enough to be used by Rational COBOL Runtime. The TWA length must be greater than or equal to the sum of 1024 plus the **twasOffset** (TWA offset) build descriptor option specified when the initial program in the transaction was generated. Use the **TWASIZE** parameter in the **TRANSACTION** definition to define a TWA with an adequate length for the transaction.
- ELA3** Load for module **ELARSCNT** was not successful. Rational COBOL Runtime has not been installed correctly. Ensure the CICS region can access the Rational COBOL Runtime library and that module **ELARSCNT** is defined in the **PROGRAM** definition.
- ELA4** Load for module **ELARPRTX** was not successful. Rational COBOL Runtime has not been installed correctly. Ensure the CICS region can access the Rational COBOL Runtime library and that module **ELARPRTX** is defined in the **PROGRAM** definition.
- ELA5** Load for module **ELARPRTC** was not successful. Rational COBOL Runtime has not been installed correctly. Ensure the CICS region can access the Rational COBOL Runtime library and that module **ELARPRTC** is defined in the **PROGRAM** definition.
- ELA6** The dynamic storage stack used for working storage for Rational COBOL Runtime modules was exhausted and Rational COBOL Runtime could not continue. This problem should not occur. Report the problem to the IBM support center.
- ELA7** A **GETMAIN** was not successful. There was not enough storage for the program to complete. Try the program again when the region is less busy or try it again in a larger region.
- ELA9** Load or link for a Rational COBOL Runtime module was not successful. Rational COBOL Runtime has not been installed correctly. Use **CEDF** to determine the module name. Look for a **PGMIDERR** on a **CICS LOAD** or **CICS LINK** command. Ensure that the CICS region can access the Rational COBOL Runtime library and the module name being loaded is defined in the **PROGRAM** definitions.
- ELAB** A call was made to a main program, which is not allowed or a non-EGL program was transferred to with a **transfer to program** statement and the **isExternal=YES** option was not specified on the **transfer to program** statement or the **EXTERNALLYDEFINED** option was not specified in the **linktype** option in the **transfer to program** entry in the linkage table part.
- ELAC** Rational COBOL Runtime has detected a **FREEMAIN** request that is not valid. Collect the dump and contact the IBM Support Center for assistance.
- ELAE** A generated program has ended because of a serious error. This occurs for one of the following reasons:
- Storage has been corrupted so that a dump is necessary to debug the abend.
 - Error handling was unable to write messages to the error destination queue or to the user at the terminal. The dump is necessary to make the

diagnostic information available. The situation can occur if the error destination queue specified for the transaction using the diagnostic controller utility is not defined to CICS. In CICS, if the error destination queue is defined as an intrapartition queue, this situation occurs when there is no more space on the intrapartition queue and the error messages cannot be written.

- A severe error has occurred. Refer to the error destination queue for the corresponding error messages. The default name is ELAD. The queue name can be changed using the diagnostic controller utility.

See “Rational COBOL Runtime ABEND Dumps” on page 151 for information on how to find error messages in the dump on an ELAE abend.

ELAF ELATSRST has detected one of the following errors:

- ELATSRST was not initiated with a CICS XCTL command (for example, the restart transaction ID was associated directly to ELATSRST).
- The COMMAREA length on entry was not 0 or 10.
- The Rational COBOL Runtime portion of the TWA had been initialized, indicating that a **converse** was not in process or the non-EGL program uses the TWA and the EGL program was not generated with the proper TWA offset.
- Information in the COMMAREA was not valid, indicating that a **converse** statement was not in process.
- Information in the COMMAREA indicates that ELATSRST was started with a **show** statement or during the **inputForm** processing for the program.

ELAW A program and its associated FormGroups or a FormGroup and its associated tables were generated using incompatible versions of COBOL generators. For example, the FormGroup might have been generated by Cross System Product and the program generated by EGL.

ELAX An exception has been detected, or thrown by the user, in part of the CICS EGL application or by a subsequently called application, that has not been handled by an EGL language **try ... onException** block. As this unhandled exception has made its way back to the main EGL program without being handled, a CICS abend of ELAX is issued. To determine the cause of this unhandled exception, the easiest way is to look in the ELAD queue under CICS by using these two commands: CEBR, and then as a response to the CEBR transaction, enter: GET ELAD. This will display the messages in the ELAD queue. These messages are ordered chronologically, so look near or at the bottom of the queue. There will be error messages about the type of exception, the program it was in, and the line number where it occurred. Alternatively, if the EGL **statementTrace** build descriptor option is set to YES, then the statement trace spool file will also show where the exception was thrown.

IMS, IMS BMP, and z/OS Batch Environments

1600 A generated program has ended because of a serious error. This occurs for one of the following reasons:

- Storage has been corrupted so that a dump is necessary to debug the abend.
- Error handling was unable to write messages to the error destination queue or to the user at the terminal. The dump is necessary to make the

diagnostic information available. In IMS, the situation can occur if the error destination queue specified using the **errorDestination** build descriptor option is not defined to IMS.

- A severe error has occurred. In IMS, refer to the error destination queue specified using the **errorDestination** build descriptor option for the corresponding error messages. In z/OS batch, refer to the data set ELAPRINT for the messages.

See “Rational COBOL Runtime ABEND Dumps” on page 151 for information on how to find error messages in the dump on a 1600 abend.

- 1601** A database manager or subsystem interface module (for example, ASMTDLI for DL/I access) was not linked with a Rational COBOL Runtime program at product installation. Registers 3 and 4 in the dump contain the name of the stub program. The abending load module is the module that was not linked correctly.

Refer to the *Program Directory for Rational COBOL Runtime for zSeries* for information on correctly linking the abending load module.

- 1602** A program generated with the **imsFastPath=YES** build descriptor option ended because of a run unit error. The abend is issued to prevent any further scheduling of the program in error.

See “Rational COBOL Runtime ABEND Dumps” on page 151 for information on how to find error messages in the dump on a 1602 abend. Depending on the build descriptor options specified for the program, the message might also have been written to an error diagnostic message queue, to the IMS log, or to an ELAPRINT file. See Chapter 18, “Diagnosing Problems for Rational COBOL Runtime on z/OS Systems” for more information on Rational COBOL Runtime error reporting.

- 1606** The dynamic storage stack used for working storage for Rational COBOL Runtime modules was exhausted and Rational COBOL Runtime could not continue.

This problem should not occur. Report the problem to the IBM Support Center.

- 1608** Rational COBOL Runtime has detected a FREEMAIN request that is not valid. Collect the dump and contact the IBM Support Center for assistance.

- 1610** A program and its associated FormGroups, or a FormGroup and its associated tables were generated using incompatible versions of COBOL generators. For example, the FormGroup might have been generated by Cross System Product and the program generated by EGL.

- 3888** An exception has been detected, or thrown by the user, in part of the EGL application or by a subsequently called application, that has not been handled by an EGL language **try ... onException** block. This exception made its way back to the main EGL program without being handled. To determine the cause of this unhandled exception, look at the output for the ELAPRINT DD statement for a z/OS Batch job, IMS BMP job, or IMSVS region JCL. This output contains error messages about the type of exception, the program it was in, and the line number where it occurred. Alternatively, if the EGL **statementTrace** build descriptor option is set to YES, then the statement trace shows where the exception was thrown.

Exception Codes

The following exception codes are issued by the Rational COBOL Runtime:

- 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/I exception (not supported for EGL Version 7.0)
- 9990 File I/O exception
- 9991 MQ I/O exception (not supported for EGL Version 7.0)
- 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 24. Codes from Other Products for z/OS Systems

The chapter contains lists of common system abend codes, COBOL runtime messages, LE abend codes, and common runtime messages and abend codes from IMS and CICS

Common System Abend Codes for All Environments

Only the most frequently occurring abend codes are listed in this section. If you receive another abend code or if you need a more complete explanation of one of the abend codes, refer to the *System Codes* manual for your release of z/OS.

System 0C4 This code can occur on a **transfer to program** statement if there is a print services or table program with the same name as the transferred-to program. This code can also occur when a print services or table program is called but there is a different program (for example, a non-EGL program or an EGL program) with the same name. Using naming conventions can eliminate this problem.

This code can also occur if you add the **validatorDataTable** property to a form in a FormGroup that is shared by multiple programs and do not generate all the programs again.

System 0C7 Data exception. The abend occurs when fields defined as NUM, NUMC, DECIMAL, or PACF are retrieved from a database or file and are found to contain data of a different format.

The abend can also occur if fields that are not initialized are used in calculations or comparisons. This happens if the program attempts to read a record from a database and the record is not found, but the program uses fields in the record anyway.

The abend can also occur if one of the following is true:

- There are redefined records with different data types or variable field boundary alignments from the original record.
- The **inputRecord** for the program receives a transferred record that contains different data types or variable-field boundary alignments from the original record.

For initialization problems with NUM and NUMC fields, you might be able to use the **spacesZero="YES"** build descriptor option to help minimize the problem. However, be sure to consider the performance implications first.

System 806 Module not found in a library. This can occur if a new version of a module is put into a load library and is placed in secondary extents. To avoid this when you allocate load libraries, specify a large primary allocation and 0 for the secondary allocation. This insures that if there is enough space for the load module it will be placed in the primary extent. If there is not enough space, there will be an abend (for example, a B37 abend for insufficient space) when you link the module into the load library. Using this technique detects the space problem during the preparation step rather than at run time.

In IMS, this abend can occur if a program transfers to another program using a **transfer to transaction** statement and the transaction named on the statement is defined in the IMS system definition, but the load module for the program is not in a library available to the IMS message region.

In other environments, this abend can occur if the module is not in a library defined in your link list, JOBLIB, or STEPLIB concatenation sequence.

If the missing module name is ELACxxx, the NLS language code identified by the last 3 characters of the module name is not installed on the system. This language code was specified with the **targetNLS** build descriptor option when the program was generated.

If you try to run an EGL-generated program under Rational COBOL Runtime and cannot load the module ELARSCNT, the system abends with an 806.

LE Runtime Messages

Only the most frequently occurring LE runtime messages are listed in this section. If you receive other runtime messages that start with IGZ or if you need a more complete explanation of one of the messages, refer to the debugging manual for your release of LE.

IGZ0033S **An attempt was made to pass a parameter address above 16 megabytes to AMODE(24) program program-name.**

Explanation: An attempt was made to pass a parameter located above the 16-megabyte storage line to a program in AMODE(24). The called program will not be able to address the parameter.

Programmer response: If the calling program is compiled with the RENT option, the DATA(24) option may be used in the calling program to make sure that its data is located in storage accessible to an AMODE(24) program. If the calling program is compiled with the NORENT option, the RMODE(24) option may be used in the calling program to make sure that its data is located in storage accessible to an AMODE(24) program. Verify that no linkedit, binder or genmod overrides are responsible for this error.

System action: The program was terminated

IGZ0064S **A recursive call to active program program-name in compilation unit compilation-unit was attempted.**

Explanation: COBOL does not allow reinvocation of an internal program which has begun execution, but has not yet terminated. For example, if internal programs A and B are siblings of a containing program, and A calls B and B calls A, this message will be issued.

Programmer response: Examine your program to

eliminate calls to active internal programs.

System action: The program was terminated.

IGZ0066S **The length of external data record data-record in program program-name did not match the existing length of the record.**

Explanation: While processing External data records during program initialization, it was determined that an External data record was previously defined in another program in the run unit, and the length of the record as specified in the current program was not the same as the previously defined length.

Programmer response: Examine the current file and ensure the External data records are specified correctly.

System action: The program was terminated.

IGZ0075S **Inconsistencies were found in EXTERNAL file file-name in program program-name. The following file attributes did not match those of the established external file: attribute-1 attribute-2 attribute-3 attribute-4 attribute-5 attribute-6 attribute-7**

Explanation: One or more attributes of an external file did not match between two programs that defined it.

Programmer response: Correct the external file. For a summary of file attributes which must match between definitions of the same external file, see the COBOL

Common COBOL Abend Codes

Only the most frequently occurring abend codes are listed in this section. If you receive another abend code or if you need a more complete explanation of one of the messages, refer to the debugging manual for your release of LE.

User 4087 This is an LE abend code. If reason code is 7, the error could be due to the region size not being large enough to run the COBOL program.

Common IMS Runtime Messages

Only the most frequently occurring IMS runtime messages are listed in this section. If you receive another runtime message that starts with DFS or if you need a more complete explanation of one of the messages, refer to the IMS messages and codes manual for your release of IMS.

DFS057I **REQUESTED BLOCK NOT AVAILABLE: blockname RC = reason code**

Explanation: The blockname is either the MOD or the DOF name. If it is the DOF name, the first 2 bytes of the name are the device type and features printed in hexadecimal. Refer to the message format services manual for your release of IMS for an interpretation of these 2 bytes. If it is a MOD name, it will be the name of a FormGroup.

User response: If a DOF name was specified, review the values you specified for the **mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore** build descriptor options, and compare them to the IMS system definition for the terminal that had the problem.

If a MOD name was specified, ensure that you installed the MFS control blocks into the correct library. If you set the **mfsUseTestLibrary** build descriptor option to YES, ensure that you used the /TEST MFS command. If you set **mfsUseTestLibrary** to NO, ensure that your system administrator has run the IMS online change utility to copy in the new format definitions.

DFS064 **NO SUCH TRANSACTION CODE**

Explanation: This message is sent to a terminal when the transaction code requested by the user is not defined to IMS. An example of a situation that results in this message is when a program uses a **show** statement to transfer to a transaction that is not defined to IMS. The form specified in the **show** statement is written to the terminal, but when the user enters data, the transferred-to transaction cannot be scheduled because it is not defined to IMS.

User response: Either ensure the transaction code is defined to IMS or change the **show** statement in the transferring program to reference the correct IMS transaction code.

DFS182 **INVALID OR MISSING PARAMETER**

Explanation: An IMS reserved word (for example, LTERM) was used as a form name in a /FORMAT command.

User response: If you need to use the /FORMAT command to display this form, you need to change the form name and generate the FormGroup and any programs that use this form again.

DFS555I **TRAN tttttttt ABEND S000,Uaaaa; MSG IN PROCESS: (up to 78 bytes of data) time stamp**

Explanation: This message indicates that the transaction running in IMS has ended abnormally. Typical abend codes are shown below:

0778 IMS user abend, indicating that a ROLL request was issued.

1602 Rational COBOL Runtime abend because a run unit error occurred in a program that was generated with the **imsFastPath="YES"** build descriptor option.

1600 Rational COBOL Runtime abend because an unrecoverable error occurred in situations other than run unit errors for programs generated with **imsFastPath="YES"**.

User response: Press the PA1 or PA2 key to display the error form that contains the error diagnostics that describe the error.

DFS2082 **RESPONSE MODE TRAN TERMINATED WITHOUT REPLY**

Explanation: Rational COBOL Runtime has ended the logical unit of work for a program that was generated with the **imsFastPath="YES"** build descriptor option.

User response: Press the PA1 key to display the error form that contains the error diagnostics that describe the error.

DFS2766I PROCESS FAILED

Explanation: IMS issues this message if Rational COBOL Runtime ends the run unit for a transaction program that was generated with `imsFastPath="YES"` and run in an IMS fast-path region.

User response: Press the PA1 or PA2 key to display the error form that contains error diagnostics that describe the error. See Chapter 18, "Diagnosing Problems for Rational COBOL Runtime on z/OS Systems" for additional information.

Explanation: One of the following might have occurred:

- The program attempted to display a form with DBCS or mixed data on a non-DBCS terminal or printer.
- The values specified for the `mfsDevice`, `mfsExtendedAttr`, and `mfsIgnore` build descriptor options do not match the IMS system definition for the terminal that had the problem.

User response: Correct the program or build descriptor options, generate the program and FormGroup again, and then run the program again.

(none) Logged off IMS and returned to the VTAM sign-on screen without any warning or error message being displayed.

Common IMS Runtime Abend Codes

Only the most frequently occurring IMS abend codes are listed in this section. If you receive another abend code or if you need a more complete explanation of one of the abend codes, refer to the messages and codes manual for your release of IMS.

IMS 259 A program has been compiled with the DATA(31) compile option and is being run in a non-IMS/ESA environment. The program should be recompiled with the DATA(24) compile option.

IMS 462 A program was scheduled in a message region, but the program ended without successfully issuing a get unique for an input message. This can occur if Rational COBOL Runtime detects an error that would prevent the program from processing properly. Examples of these errors are:

- The IMS PSB does not match the EGL PSB record definition.
- The print services program is missing.

IMS 778 A ROLL call has been issued by Rational COBOL Runtime because of a run unit error or a catastrophic error in the IMS/VS environment. The ROLL is issued to prevent further scheduling of the program in error. IMS displays message DFS555I indicating that abend 778 has occurred. The Rational COBOL Runtime error message panel can be displayed by pressing PA1.

Based on your build descriptor options and the JCL for your message region, additional diagnostic information might be provided on an error diagnostic message queue, in the IMS log, or in ELAPRINT. See "Controlling Error Reporting in IMS Environments" on page 140 for additional information.

Note: Press PA2 if PA1 does not cause the Rational COBOL Runtime error form to display.

IMS 1008 A program that was running as a BMP and that obtained access to fast-path databases did not issue a SYNC or CHKP call at the end of the job step. You can force the CHKP call to occur by:

- Using the **sysLib.commit()** system function in a batch-oriented BMP
- Ensuring that the transaction-oriented BMP ends with an **endOfFile** (QC status) for the file being used for input from the IMS message queue

IMS 3042

Access to DB2 cannot be obtained. Possible causes of this are:

- The terminal ID is not defined to DB2.
- The DB2 plan is not valid or access to the DB2 plan cannot be obtained.

If the program was being run as a BMP, see Figure 21 on page 115 for sample JCL.

Common CICS Runtime Messages

Only the most frequently occurring CICS runtime messages are listed in this section. If you receive another CICS runtime message that starts with DFH or if you need a more complete explanation of one of the messages, refer to the CICS messages and codes manual for your release of CICS.

DFHAC2016 date time **applied Transaction** tranid
cannot run because program
program-name is **not available**.

Explanation: The transaction tranid cannot be run because the initial program for the transaction is not available. This could occur because the transaction is defined, but the program is not defined or is not in a library in the DFHRPL concatenation.

User response: Have your system administrator check the RDO PROGRAM entries or ensure that CICS autoinstall is enabled for programs. Be sure the program is in a library in the DFHRPL concatenation.

DFHAC2206 time applied **Transaction** tranid **has failed with abend** abcode. **Resource backout was successful**.

Explanation: The transaction tranid has ended abnormally with abend code abcode. abcode is either an CICS transaction abend code or a user abend code.

User response: If the user abend code starts with ELA, see “CICS Environments” on page 185. If it is an CICS abend code, see “Common CICS Abend Codes” to see if it is included there. If not, refer to the CICS messages and codes manual for your release of CICS.

Common CICS Abend Codes

Only the most frequently occurring CICS abend codes are listed in this section. If you receive another CICS abend or if you need a more complete explanation of one of the abend codes, refer to the CICS messages and codes manual for your release of CICS.

Depending on your diagnostic control options, information might be available on an error destination queue or in an CICS journal. For more information, see “Controlling Error Reporting in CICS” on page 140.

- ADCA** An error occurred while processing a DL/I request. In addition to looking for the information provided by CICS, look for messages or abends from DL/I.
- ADLD** A program isolation deadlock occurred and a transaction was selected for an abend. For information on using the **dliVar.cicsRestart** system variable, or for information on designing restartable transactions, see the *EGL Language Reference*.
- AEY9** Access to DB2 cannot be obtained. This occurs if DB2 is not running.
- AFCY** A transaction was purged when a deadlock occurred because a file is defined with LSRPOOLID not equal to NONE in the FCT, and

one function within a program has performed a **get next** against a file and another function requested an update or add to the same file (or its alternate index) without ending the **get next**. Change the LSRPOOLID to NONE, or change the program design to end the **get next** before the update or add is requested.

APCT	A requested module cannot be located in the program definitions or in the program library.
ASRA	<p>A program check occurred. Some of the reasons this can occur for an EGL program are as follows:</p> <ul style="list-style-type: none">• Incorrectly linked Rational COBOL Runtime modules. If register 1 contains ELA1, see the information for ELA1 in "CICS Environments" on page 185.• Data not initialized or data initialized to incorrect values. If the error occurred as a result of a data exception, see the explanation for "System 0C7" in "Common System Abend Codes for All Environments" on page 191.
ATDD	The program attempted to process a transient data queue that is disabled. This can occur for a program file associated with a transient data queue or for the transient data queue used for error diagnostic information.
AXFQ	The most common cause is the result of INBFMH not being specified equal to ALL in the profile associated with the CICS mirror program (CPMI).

Note: CICS users that receive abend codes ADLD, ADCP, AKCT, or D106 might see four question marks in place of the CICS abend code for the resulting Rational COBOL Runtime message. The CSMT console log contains the true CICS abend code that was issued.

COBOL Abends under CICS

- 1009** A program has a dynamic storage requirement greater than 64KB, but was compiled with the DATA(24) compiler option. Compile the module again with the DATA(31) compiler option.
- 1029** One of the following situations occurred:
- A PROGRAM entry for a program attached through a COBOL dynamic call is not found and CICS autoinstall is not enabled for programs
 - The module being invoked cannot be found in the CICS region program library search string

Additional information can be retrieved by entering transaction CEBR on the terminal where the error occurred.

Part 6. Appendixes

Appendix. Rational COBOL Runtime Messages

This section describes a series of messages that are given by Rational COBOL Runtime.

Message Format

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

The message identifier consists of three fields: prefix, message number, and type code. The format of the message identifier is *xxxnnnnnt*, where:

xxx Message prefix, as follows:

- ELA** These runtime messages can occur when your program stops, ends with an error, or requires special attention.
- FZE** These runtime messages can occur when using the installation and print utilities FZEZREBO and FZETPRT that are provided with Rational COBOL Runtime
- PRM** These messages can occur when you are using the parameter group utility.

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

t Type code, as follows:

I Information

Indicates a minor error, such as a move from a field that is not initialized, or provides you with general information about the process you are working on. Processing continues

A Action

Indicates that you must take some specific action before the process can continue (for example, a YES or NO response might be required). Processing continues after you complete the required action.

P Problem Determination

Indicates that a problem condition exists that requires diagnosis. Processing ends when this type of message is issued. If the problem determination message text includes a return code, see Chapter 22, "Common System Error Codes for z/OS Systems," on page 167 for an explanation of the return code:

S System Action

Indicates that a system error occurred requiring you to take some action. These messages appear in English.

The message text might contain one or more inserts. When the message is displayed an insert is used to fill in names, constants, return codes, and so forth. The format of the message insert is *%*xy*zz*, where:

xx Number of the insert

y C, D, or X. These letters represent the following:

- C Characters (usually a name)
- D Decimal numbers (usually a length, record count, or error count)
- X Hexadecimal numbers (usually a return code)
- zz Length of the insert

In this manual you see messages listed like this:

```
ELA00023P Call to DataTable program %01C07 was not successful
```

If you receive this message on your system, the insert is automatically converted. For example, if there is a problem with DataTable program TABLNAM, the error is displayed on your system like this:

```
ELA00023P Call to DataTable program TABLNAM was not successful
```

TABLNAM is the first insert of the message (%01) and is in character format (C) and is seven characters long (7).

ELA Messages

ELA00002P IBM Rational COBOL Runtime is required for program %01C08.

Explanation: The generated COBOL program is not compatible with the installed version of Rational COBOL Runtime.

Rational COBOL Runtime ends the program with a userabend.

User response: Verify that the latest maintenance has been applied. You can find the latest maintenance level in the technote located at the following website:

<http://www-01.ibm.com/support/docview.wss?uid=swg21444221>

If the maintenance level is current, verify the maintenance is applied to the correct load module library. To determine the maintenance level, locate the ELARSCNT load module and scroll to the right of the module name to view the PTF associated with the module. PTF numbers begin with UK. Make sure that you are looking at the correct load module.

- If the PTF number does not match the minimum required PTF, install at least the minimum required PTF level.
- If the PTF number matches the minimum required PTF level:
 - Verify that you are pointing to the correct SELALMD load module library
 - For CICS: Verify that the ELARSCNT load module that the maintenance has been applied to is the same module that is being used by CICS. Note that a new copy must be created to pick up new load module versions.

%02C04, status code = %03C02

Explanation: The program control logic attempted a DL/I call to a teleprocessing PCB and received an error status code from IMS on the call. The message specifies the PCB that was used on the call (0 is the I/O PCB, 1 is the modifiable alternate PCB, and 2 is the express modifiable alternate PCB). The message also specifies the function code and the status code. For ISRT calls, the message is accompanied by message ELA00066I, which displays the first 255 bytes of the DL/I I/O area.

The run unit ends. If the ELASNAP data set is allocated, Rational COBOL Runtime issues a SNAP dump for all status codes other than AI.

User response: Look up the status code in the IMS messages and codes documentation for your system.

ELA00005A Date entered is not valid for defined date format %01C10

Explanation: Data entered into a form field defined with a **dateFormat** property either 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 they are omitted, enter leading zeros. For example, if the date format is MM/DD/YY, you can enter 070491.

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

ELA00007P File OPEN error on file %01C08, file status = %02C08

Explanation: The specified file did not open successfully.

ELA00003P PCB %01D03 DL/I error, function =

The format of the file status depends on the file type.

For SEQ files, the file status is the 2-character COBOL status code followed by six zeros.

For VSAM files, the file status is composed of the 2-character COBOL status code followed by the VSAM return code (two characters), VSAM function code (one character), and the VSAM feedback code (three characters). The VSAM codes could be blank if the file OPEN was not completed.

For VSAMRS files, the file status is composed of the 2-character ACB (access control block) return code in hexadecimal format followed by six zeros.

The run unit ends.

User response: First see the tables of common COBOL and VSAM status codes in the Chapter 22, "Common System Error Codes for z/OS Systems," on page 167. If the codes in the message are not listed in the tables, refer to the COBOL programming language reference and VSAM administration guide for your system for a definition of other file status and VSAM codes. Also look for system error messages pertaining to the specified DD name or DLBL name. Correct the error and run the program again.

ELA00008P File CLOSE error on file %01C08, file status = %02C08

Explanation: The specified file did not close successfully, and the run unit ends.

The format of the file status depends on the file type.

For SEQ files, the file status is the 2-character COBOL status code followed by six zeros.

For VSAM files, the file status is composed of the 2-character COBOL status code followed by the VSAM return code (two characters), VSAM function code (one character), and the VSAM feedback code (three characters).

For VSAMRS files, the file status is composed of the 2-character ACB (access control block) return code in hexadecimal format followed by six zeros.

The run unit ends.

User response: First see the table of common COBOL and VSAM status codes in the Chapter 22, "Common System Error Codes for z/OS Systems," on page 167. If the codes in the message are not listed in the tables, refer to the COBOL programming language reference and VSAM administration guide for your system for a definition of other file status and VSAM codes. Also look for system error messages pertaining to the DD name. Correct the error and run the program again.

ELA00009P 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 does not handle the overflow exception that occurred, then the program ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Have the application developer do one of the following:

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

ELA00014P A replace was attempted without a preceding get for update on %01C18

Explanation: A **replace** was attempted for a record that has not been successfully read by a **get forUpdate** or an **open forUpdate** statement. The read for update might have been lost as the result of a commit or rollback or as the result of a **converse** statement in a segmented program.

The run unit ends.

User response: Ensure that the **replace** statement and the corresponding **get forUpdate** or **open forUpdate** correctly use the same record variable name or resultSetID.

Also make sure that the sequence of statements is appropriate. To step through the program, you can use the EGL debugger or (for CICS-based programs) CEDF.

ELA00015P READ/WRITE error for file %01C08, file status = %02C08

Explanation: An I/O operation was not successful for the specified file. Program processing ends on any nonzero status code if the I/O statement is not in a **try** block; and ends on a hard error if the I/O statement is in a **try** block when **vgVar.handleHardIOErrors** is set to 0.

The format of the file status depends on the file type.

For SEQ files, the file status is the 2-character COBOL status code followed by six zeros.

For VSAM files, the file status is composed of the 2-character COBOL status code followed by the VSAM return code (two characters), VSAM function code (one character), and the VSAM feedback code (three characters).

The run unit ends.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: First see the tables of common COBOL and VSAM status codes in the Chapter 22, "Common System Error Codes for z/OS Systems," on page 167. If the codes in the message are not listed in the tables, refer to the COBOL programming language reference and VSAM administration guide for your system for a definition of the other file status and VSAM codes. Also look for system error messages pertaining to the specified DD name. Correct the error and run the program again.

ELA00016P %01C08 error for file %02C08, %03C44,
file status = %04C08

Explanation: An I/O operation was not successful for the specified file. Program processing ends on any nonzero status code if the I/O statement is not in a **try** block; and ends on a hard error if the I/O statement is in a **try** block when **vgVar.handleHardIOErrors** is set to 0.

The message identifies the VSAM operation that was not successful, the EGL file name associated with the record, the system resource name, and the file status. The file status is composed of two zeros followed by the VSAM return code (two characters), VSAM function code (one character), and the VSAM feedback code (three characters).

The run unit ends.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: First see the tables of common VSAM status codes in the Chapter 22, "Common System Error Codes for z/OS Systems," on page 167. If the codes in the message are not listed in the tables, refer to the VSAM administration guide for your system for a definition of other VSAM codes. Also look for system error messages pertaining to the specified system resource. Correct the error and run the program again.

ELA00021I An error occurred in program %01C08
on statement number %02D06

Explanation: An error occurred in the specified program on the specified statement. The actual error

that occurred is identified in the messages following this message.

User response: Correct the statement, and generate the program again.

ELA00022P Form group format module %01C08
could not be loaded

Explanation: The specified FormGroup format module could not be loaded. The module is a generated object module linked as a program that contains tables that describe the format and constant fields for text forms in a FormGroup. The module name is the FormGroup alias (or a variation to conform with length and character restrictions) followed by the characters FM.

If the format module name uses the format ELAxxxFM, where xxx is the language code, the definitions for the Rational COBOL Runtime error forms could not be loaded.

The run unit ends.

User response: Make sure that the specified program was generated, compiled, and linked into a library defined in the library search order.

For z/OS CICS, the search order includes the DFHRPL data sets, and you should verify that the program has been defined to the system.

For IMS/VS environments, the search order includes the STEPLIB and JOBLIB data sets

ELA00023P Call to DataTable program %01C08 was
not successful

Explanation: A dynamic COBOL call to the specified DataTable program was not successful. The run unit ends.

User response: Make sure that the specified program was generated, compiled, and linked into a library defined in the library search order.

For z/OS CICS, the search order includes the DFHRPL data sets. Verify that the program has been defined to the system. Also ensure that the program was generated with the **data="31"** build descriptor option.

For IMS/VS, IMS BMP, or z/OS batch, the search order includes the STEPLIB and JOBLIB data sets.

If the program named in the messages is ELACxxx or ELAYYNx (where xxx and x are the NLS identifiers), verify that the customization JCL in job ELACJxxx has been run. Also verify that the appropriate language (indicated by xxx or x) has been installed.

ELA00024P Conversion table %01C08 could not be loaded

Explanation: Either the specified conversion table program could not be loaded or the program that was loaded is not a Rational COBOL Runtime conversion table.

The run unit ends.

User response: Verify that the correct conversion table name was specified in the generation-time linkage options part; that a correct conversion table has been moved into the system variable `sysVar.callConversionTable` at run time; or that a correct conversion table has been specified when using the `sysLib.convert()` system function. For more information, see "callConversionTable" in the EGL online help system.

If the conversion table was properly specified in the program, make sure that the table program was generated, compiled, and linked into a library defined in the library search order.

For z/OS CICS, the search order includes the DFHRPL data sets. Verify that the program has been defined to the system. Also ensure that the program was generated with the `data="31"` build descriptor option.

For IMS/VS, IMS BMP, or z/OS batch, the search order includes the STEPLIB and JOBLIB data sets.

If the conversion table program is defined in the load library, verify that the program is using either a conversion table shipped with Rational COBOL Runtime or a table created using the conversion table format. For information on creating a custom conversion table, see "Creating a custom conversion table" on page 22. For more information on conversion tables in general, see "Data conversion" in the *EGL Generation Guide*.

ELA00026P A calculation caused a maximum-value overflow

Explanation: During a calculation, an intermediate result exceeded the maximum value. 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 division by zero occurs. 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, then the program ends.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime

issues a SNAP dump if the ELASNAP data set is allocated.

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.

ELA00027P The data on a 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.

The run unit ends.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

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

ELA00029P Transfer to %01C08 was not successful

Explanation: The transfer to another program was not successful. Usually, the program being transferred to could not be found.

The run unit ends.

User response: Make sure that the program was generated, compiled, and linked into a library defined in the library search order.

For z/OS CICS, the search order includes the DFHRPL data sets. Verify that the program has been defined to the system. Also ensure that the program was generated with the `data="31"` build descriptor option.

For IMS/VS, IMS BMP, or z/OS batch, the search order includes the STEPLIB and JOBLIB data sets.

ELA00031P Call to %01C08 was not successful

Explanation: A dynamic call to the specified program failed, ending the run unit.

User response: Make sure that the program was generated, compiled, and linked into a library defined in the library search order.

For z/OS CICS, the search order includes the DFHRPL data sets. Verify that the program has been defined to the system. Also ensure that the program was

generated with the **data="31"** build descriptor option.

For IMS/VS, IMS BMP, or z/OS batch, the search order includes the STEPLIB and JOBLIB data sets.

ELA00032P Called program %01C08 received a parameter list that is not valid

Explanation: A call to the specified program was not successful for one of the following reasons:

- The calling program passed too many or too few parameters.
- Different values are in the linkage options part, **callLink** element, **parmform** property for the called and calling programs.
- The **parmform** value COMMDATA was specified for the call, and the COMMAREA passed has a different length than the length expected by the called program.

If the called program is a remote program running on CICS, a CICS abend occurs. Because the COMMAREA is too small, the called program cannot notify the calling program of the error.

In all other cases, the run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Verify that the argument list in the **call** statement matches the parameter list for the program being called, and then generate the called and calling program with the same **parmform** value for the **callLink** element of the linkage options part.

ELA00033P Call to program %01C08 returned exception code %02D05.

Explanation: An exception code was returned on a call to the specified program, indicating that one of the arguments passed to the program was not valid. The run unit ended because the call was not in a **try** block.

User response: Place the **call** statement in a **try** block and make sure that all the passed arguments are valid.

ELA00034P Program %01C08 was declared as a main program and cannot be called

Explanation: The specified program was not declared as a called program.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Declare the program as a called program.

ELA00035A 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.

ELA00036A 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.

ELA00037A Input not within defined range - enter again

Explanation: The data in the first highlighted field is not within the range of valid data 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.

ELA00038A Table edit validity error - enter again

Explanation: The data in the first highlighted field does not meet the **validatorDataTable** requirement defined for the variable field.

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

ELA00039A Modulus check error on input - enter again

Explanation: The data in the first highlighted field does not meet the modulus check 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.

ELA00040A No input received for required field - enter again

Explanation: No data was typed in the field designated by the cursor. 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 do not satisfy the data input requirement for any type of field. In addition, zeros do not satisfy the data input requirement for numeric fields. The program continues.

ELA00041P Property msgTablePrefix was not specified for a program: Message %01C04, NLS code %02C03

Explanation: The program tried to display a message from the message table using the `converseLib.validationFailed()` system function. However, the program does not specify a value for the `msgTablePrefix` property.

The run unit ends.

User response: Do any of the following:

- 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 then generate the program again.
- Remove the user message number from the form field message properties and generate the program and FormGroup again.

ELA00042P The expected number of inserts for message %01C08, NLS code %02C03 was not received

Explanation: The expected number of variable inserts for an Rational COBOL Runtime message did not match the number received. The message text is in the language-dependent message DataTable program, ELACxxx, where xxx is the language code.

The inserts show the original error message number that occurred and the language code being used. Message ELA00163P shows the original error message number that occurred and the message inserts that would have been displayed for that message.

The run unit ends.

User response: Correct the problem identified by the original message.

If the language-dependent message DataTable was modified, correct the modified message so that the inserts are the same as the inserts defined in the default message DataTable that was shipped with Rational COBOL Runtime.

ELA00043P %01C08, %02C03

Explanation: The Rational COBOL Runtime message DataTable program ELACxxx (where xxx is the language code) did not contain a runtime message.

The inserts show the original error message number that occurred and the language code being used. Message ELA00163P shows the original error message number that occurred and the message inserts that would have been displayed for that message.

The run unit ends.

User response: Correct the problem identified by the original message.

If the language-dependent message DataTable was modified, verify that the message numbers in the modified DataTable match the message numbers in the message DataTable as shipped in the product. Also, verify that the program loaded is at the same maintenance and release level as the default message DataTable shipped with Rational COBOL Runtime.

ELA00044P Message %01C08, NLS code %02C03, not found

Explanation: The Rational COBOL Runtime message DataTable program ELACxxx (where xxx is the NLS code) did not contain a runtime message.

The inserts show the original error message number that occurred and the NLS language code that was being used. The message is accompanied by message ELA00163P, which shows the original error message number that occurred and the message inserts that would have been displayed for that message.

The original error message that occurred determines if (and how) the program ends and if a SNAP dump is issued.

User response: Correct the error identified by the first message insert.

If the message DataTable was modified, check that the message numbers in the modified DataTable match the message numbers in the default message DataTable shipped with Rational COBOL Runtime. Also, check that the program loaded is at the same maintenance and release level as the default message DataTable shipped with Rational COBOL Runtime.

ELA00045P Error reading message %01C08, NLS code %02C03, status %03C08

Explanation: The user message file or database did not contain a user-defined message for the language associated with the language code. Message files and databases are used only in COBOL programs generated using CSP/370 Runtime Services Version 1 Release 1.

The format of the message ID is as follows:

- Positions 1-3 = User message file

- Positions 4-8 = Message number

The status code varies depending on the type of user message file or database being used:

- For VSAM, status is eight characters. The first two bytes of code are either 08 (to specify a relative message within a record is not used) or 12 (to specify a record was not found in the VSAM file). The remaining six bytes of the status code are the VSAM return code (two characters), function (one character), and feedback code (three characters), all in decimal format. Refer to the VSAM administration guide for your system for a definition of the VSAM codes.
- For DB2, status is the 4-character SQL code. Refer to the DB2 manuals for your system for a description of the SQL code.
- For DL/I, status is the 2-character DL/I status code. Refer to the IMS messages and codes manual for your system for a description of the specified status code.
- In the IMS/VS environment, the transaction (logical unit of work) ends and processing continues with the next message.

In all other environments, the run unit ends.

User response: Make sure that the message is defined in the program message file in one of two ways:

- Convert the message file to an EGL message DataTable. Generate the program and the message DataTable again using EGL COBOL generation.
- If a message database is being used, add or replace the message in the message database using the Cross System Product/370 Runtime Services Version 1 Release 1 message database utility.

ELA00046P Call to print services program %01C08 was not successful

Explanation: A dynamic COBOL call to the specified print services program was not successful.

The run unit ends.

User response: Make sure that the program was generated, compiled, and linked into a library defined in the library search order.

For z/OS CICS, the search order includes the DFHRPL data sets. Verify that the program has been defined to the system. Also ensure that the program was generated with the **data="31"** build descriptor option. In addition, verify that the customization job, ELACJIC has been run.

For IMS/VS, IMS BMP, or z/OS batch, the search order includes the STEPLIB and JOBLIB data sets.

ELA00047P Message %01D04 was not found in message table program %02C07

Explanation: A user message could not be found in the program message DataTable.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

The run unit ends.

User response: Either add the message to the DataTable or modify the program to use a message that is defined in the message DataTable.

ELA00050A Significant digits for field exceeded - enter again

Explanation: The user entered data into a numeric field that was defined with decimal places, a sign, currency symbol, or numeric separator edits. 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. 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.

ELA00051P Form %01C08 was not found in FormGroup %02C06

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

The run unit ends.

User response: Generate the FormGroup and the program again.

ELA00057P Delete attempted without preceding update on record %01C18

Explanation: This error occurs in these cases:

- A **delete** statement was issued against a record that was not successfully read for update
- A **delete** statement is associated with a specific **get** statement, but a different **get** statement was used to select the record.

The read for update might have been cancelled as the result of a **converse** statement in a segmented program.

The run unit ends.

User response: Make sure that in the **get**, **open**, and **delete** statements, the program correctly used record names or a resultSetID.

Also make sure that the sequence of statements is appropriate. To step through the program, you can use

the EGL debugger or (for CICS-based programs) CEDF.

ELA00061P DL/I error, function = %01C04, status code = %02C02

Explanation: DL/I returned a status code in response to the DL/I call for the current I/O statement and either of the following occurred:

- There was no error routine specified for the I/O statement.
- Both **VGVar.handleHardIOErrors** and **dliVar.handleHardDLIErrors** were set to 0 (this indicates that the program should end on abnormal DL/I conditions), and the status code specified either an abnormal condition, or a condition that was not expected.

The status code in the message comes from the DL/I PCB used for the DL/I call.

The run unit ends.

In CICS environments Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

This is either a program error or a database definition error.

User response: Do the following:

1. Locate the specified error code. Refer to the IMS messages and codes manual for a description of the specified status code.
2. Correct the error.
3. Generate the program again.

ELA00062P DL/I call overlaid storage area, record %01C18

Explanation: A DL/I call read a block of data that was larger than the record defined to hold the data. The storage area immediately following the record buffer was overlaid.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: This is a program error. Define the record so that its length matches the length of the segment it represents and generate the program again.

ELA00063I PCB DB %01C08, segment %02C08, level %03D02, options %04C04

Explanation: This message provides additional diagnostic information for a database I/O error. The PCB passed in the DL/I call contained the specified information.

For unsuccessful DL/I I/O call, the segment name field contains the last segment along with the path to the requested segment that satisfied the call. When a program is initially scheduled, the name of the database might be put in the segment name field if no segment is satisfied.

User response: Refer to message ELA00061P.

ELA00064I PCB key feedback area length %01D04

Explanation: This message provides additional diagnostic information for a database I/O error. The PCB passed in the DL/I call contained the specified key feedback length. This is the length of the concatenated key of the hierarchical database path.

User response: Refer to message ELA00061P.

ELA00065I PCB key feedback area = %01C255

Explanation: This message provides additional diagnostic information for a database I/O error. The PCB passed in the DL/I call contained the specified key feedback area.

The first 255 bytes are displayed. If necessary, because of the line and data lengths, the message wraps around to display all 255 bytes. The data is displayed as character data in the message. The message is followed by two lines that give the hexadecimal value under each character.

User response: Refer to message ELA00061P.

ELA00066I DL/I I/O area = %01C255

Explanation: This message provides additional diagnostic information for a hard DL/I I/O error. The message displays the contents of the DL/I I/O area.

The first 255 bytes are displayed. If necessary, because of the line and data lengths, the message wraps around to display all 255 bytes. The data is displayed as character data in the message. The message is followed by two lines that give the hexadecimal value under each character.

User response: This message is always accompanied by another message (for example, ELA00003P or ELA00061P) that specifies the error. See the explanation and user response of the accompanying message.

ELA00067I DL/I SSA %01D02: %02C255

Explanation: This message provides additional diagnostic information for a DL/I I/O error. The message displays the contents of a segment search argument (SSA) for the DL/I call. The first message insert gives the number of the SSA. The second insert gives the first 255 bytes of the SSA.

If necessary, because of the line and data lengths, the message wraps around to display all 255 bytes. The data is displayed as character data in the message. The message is followed by two lines that give the hexadecimal value under each character.

This message is repeated once for each SSA used in the DL/I call.

User response: Refer to message ELA00061P.

ELA00068P DL/I variable segment length is not valid, segment %01C08

Explanation: A DL/I segment I/O area is shorter than the segment returned in a DL/I retrieval, or the computed segment length on an **add** or **replace** statement is not valid.

In the case of a **get**, **get forUpdate**, or **get next** statement, the BYTES parameter in the DBD is greater than the length of the record defined to EGL.

In the case of an **add** or **replace** statement, the program has erroneously set the length of the segment. If this error occurs for a path call, the DL/I I/O area shown in message ELA00061I contains only segments before the segment with the error. Because the length is in error, the segment with the error cannot be moved to the DL/I I/O area.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: If the error occurred in a retrieval, have the database administrator correct either the DBD or the EGL record definition, and generate the program again.

If the error occurred on an update, correct the logic associated with calculating the length of the segment. Generate the program again.

ELA00069P The value of an input variable is too large for the target SQL column

Explanation: When running in VisualAge Generator compatibility mode, a DECIMAL or PACF field in an SQL record is defined as requiring an even-numbered

length for SQL purposes, but has a value that is too large to be contained within the even-numbered length.

In the IMS/VS environment, the transaction (logical unit of work) ends and processing continues with the next message.

In all other environments, the run unit ends

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Modify the program to ensure that values that overflow the even-numbered length of the field are detected and rectified before executing any I/O statement that uses the SQL record, and that uses the field as an input host variable in its SQL statement.

This condition is not detected in programs that have the **checkNumericOverflow** build descriptor option set to YES; instead the high-order digit of the value of the field is truncated before being used in the SQL statement.

ELA00070P %01C04 error, status code %02C02

Explanation: DL/I returned a status code other than QC or AL in response to a CHKP (checkpoint) or ROLB (rollback) DL/I call.

CHKP and ROLB calls are issued for the following reasons:

- The program invokes the **sysLib.commit()** or **sysLib.rollback()** system functions.
- The program ends abnormally and a PSB is active.
- The program causes a commit to be taken at a **converse** statement, when reading an **inputForm**, or because the **synchOnTrxTransfer** build descriptor option is set to YES.

The status code in the message is taken from the I/O PCB used with the DL/I call.

The run unit ends.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Make a note of the message and notify the system programmer. On z/OS systems, refer to the IMS messages and codes manual for a description of the status code.

ELA00072P %01C18, set record position not supported

Explanation: The set position indicator was on for a DL/I segment record when a **get next** statement with a user-modified SSA list was used with that record. The set position indicator is not supported for DL/I calls with modified SSA lists.

The run unit ends.

User response: Modify the program logic so that it does not set the set position indicator for a segment with a modified DL/I call.

ELA00073P SQL error, command = %01C08, SQL code = %02D04

Explanation: The SQL database manager returned an error code for an SQL I/O statement. Program processing ends following an SQL request whenever the SQLCODE in the SQL communications area (SQLCA) is not 0, and either of the following is true:

- The I/O statement is not in a **try** block.
- The SQLCODE indicated a hard error and the system variable **vgVar.handleHardIOErrors** was set to 0.

The message is followed by message ELA00074I which displays the substitution variables associated with the SQLCODE. (Those substitution variables are also available to the program by way of the system variable **sysVar.sqlData**.)

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Determine the cause of the problem from the SQL code and the SQL error information.

Either correct the program or the database definition. Refer to the appropriate database manager messages and codes manual for information on the SQL code and SQL error information.

ELA00074I SQL error message: %01C70

Explanation: This message accompanies message ELA00073P when an SQL error occurs. It displays the relational database manager error information returned in the SQLCA field SQLERRM and is repeated as many times as necessary to display the complete description.

User response: Use the information from this message and ELA00073P to correct the error.

ELA00076P 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 all must occur in the following set for the move or compare to complete successfully:

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.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Change the program to ensure 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 ensure that user input contains only valid characters.

ELA00080A Hexadecimal data is not valid

Explanation: The data in the variable field identified by the cursor must be in hexadecimal format. One or more of the characters 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.

ELA00086P %01C18 - No active open or get for update is in effect

Explanation: One of these cases applies:

- A **get next** statement cannot run because a related **open** statement did not run previously in the same program; or
- A **replace** or **delete** statement cannot run because a related **open**, **get for update**, or **get next for update** 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.

The run unit ends.

User response: Make sure that in the second statement (**get next**, **replace**, or **delete**), the program correctly used a record variable name or resultSetID to match the first statement (**open** or **get**).

Also make sure that the sequence of statements is appropriate. To step through the program, you can use the EGL debugger or (for CICS-based programs) CEDF.

ELA00093I An error occurred in program %01C08, function %02C18

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

If a function is not active, the second insert contains the name of a section in the generated initialization or ending logic of the program.

User response: Refer to the error messages following this message to determine the cause of the error.

ELA00096P A data operand of type MBCHAR is not valid

Explanation: An operand in a **move** or **assignment** statement contains mixed double-byte and single-byte data that is not valid.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Verify that the data in MBCHAR variables is valid before using the variable in a **move** or **assignment** statement.

ELA00105I Error occurred at terminal %01C08, date %02C08, time %03C08, user %04C08

Explanation: An error occurred at the specified logical terminal on the specified date and time. This message precedes any error diagnostic information routed to an alternate error destination.

For a program running in z/OS batch environment, the first insert is *********, which indicates that the terminal identifier is not known.

For a batch program running in the IMS BMP or IMS/VS environments, the first variable insert is ********* if the input message queue has not yet been accessed, indicating that the terminal identifier is not known.

For the IMS BMP or z/OS batch environments, the last

insert (user) is the job name from the JOB statement in the JCL used to run the program.

For z/OS CICS and IMS/VS environments, the last insert is only provided if sign-on security is active on or provided in the system.

User response: Examine all error messages that follow this message and precede the next occurrence of this message. Use the information from these messages to diagnose and correct the error.

ELA00106P Program %01C08 PSB does not match Enterprise Generation Language PSB definition

Explanation: The PCBs passed to the program at program initialization time did not match the EGL PSBRecord defined for the program. The number of PCBs passed was less than the number of PCBs defined in the EGL PSBRecord definition.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, the Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Either correct the EGL PSBRecord definition and generate the program again, or correct the IMS PSB and generate it again.

ELA00109P Input form must be form %01C08 rather than form %02C08, for program %03C08

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

For the CICS environment, when another program transfers to this program using the **show** statement, the transferring program must specify the correct form name on the **show** statement.

For the IMS/VS environment, the initial message processed for the program must be the message input descriptor (MID) for the first identified form. Instead, the second identified form was received. Either another program transferred to this program with the wrong form, or the user did not use the **/FORMAT** command to start the program.

The run unit ends.

User response: If the error occurred when the program was started in the IMS environment, start the program using the **/FORMAT** command. Otherwise, ensure 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.

ELA00110P Shared DataTable %01C07 cannot be updated

Explanation: The program modifies a DataTable that was defined with the **shared** property set to YES. Shared DataTables cannot be updated.

The run unit ends.

User response: Either set the **shared** property for the DataTable to NO, or change the program to avoid modifying the DataTable.

ELA00111P Length of input form %01C08 is not valid

Explanation: The length of an input form received by a program is not the length defined for the form in the program.

The run unit ends.

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.

ELA00114P A transfer to called program %01C08 is not allowed

Explanation: A program cannot transfer to a called program.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility. For CICS, this message can also occur if the Rational COBOL Runtime program ELATSRTS has been used to initiate a called program.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Replace the **transfer to program** statement with a **call** statement.

ELA00115P Use of a transfer statement is invalid because the receiving program (%01C08) has an input form

Explanation: Only a **show** statement can transfer to a program that requires an input form.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Do either of these actions:

- Use a **show** statement to invoke the receiving program indirectly
- Remove the **inputForm** property of the receiving program. The program can converse the form after receiving control.

ELA00118P Missing PSB for program %01C08

Explanation: An EGL PSB was specified for the named program during definition. However, the program ran as a z/OS batch job without specifying the PSB parameter. This can happen if you do not use the sample JCL created by EGL COBOL generation.

The run unit ends.

User response: If the program contains DL/I I/O or other DL/I functions, change the runtime JCL to run DL/I programs. If the program does not use DL/I, remove the PSB name from the program definition.

ELA00119P Programs %01C07 and %02C07 are not compatible

Explanation: A program started by a **transfer to program** or **call** statement is not compatible with the initial program in the transaction or job for one of the following reasons:

- The program was generated for a different environment.
- The program is a main Text UI program, and the initial program is a main basic program (IMS/VS only).
- The programs are both main Text UI programs, but the **spaSize**, **spaADF**, or **spaStatusBytePosition** build descriptor options specified at generation are different (IMS/VS only).

The run unit ends.

User response: Change one or both programs to conform to the restrictions for a **transfer to program** or **call** statement.

ELA00120P sysLib.startTransaction not successful, logical LTERM = %01C08, status code = %02C02

Explanation: Common IMS status codes are as follows:

- QH** Unknown output destination
A1 Unknown output destination

Both status codes indicate that the 8-character logical terminal ID was not defined to the IMS system as either a terminal or transaction.

The run unit ends.

User response: Follow these steps to correct the problem:

1. Make sure that the transaction code field of the record specified in `vgLib.startTransaction()` is defined to the IMS system.
2. Review the program logic ensure that the transaction code field is set correctly.
3. Refer to the IMS messages and codes manual for your system for an explanation of status codes other than the ones listed above.

ELA00121P `sysLib.audit` was not successful, logical LTERM = %01C08, status code=%02C04

Explanation: The status code is the 2-character status from the I/O PCB.

The run unit ends.

User response: Refer to the IMS messages and codes manual for your system.

ELA00122P PCB for `dliLib.AIBTDLI`, `dliLib.EGLTDLI`, or `VGLib.VGTDLI` call not available

Explanation: The meaning varies depending on the system function as follows:

- If the system function is `dliLib.AIBTDLI`, the EGL PCB name is not associated with any PCB in the PSB being used by the program.
- If the system function is `dliLib.EGLTDLI`, the EGL PCB name is associated with a PCB number that either exceeds the number of PCBs in the PSB being used by the program or references a PCB that was not passed to the program in the called parameter list.
- If the system function is `VGLib.VGTDLI`, the EGL PCB number either exceeds the number of PCBs in the PSB being used by the program or references a PCB that was not passed to the program in the called parameter list.

The error can also occur in the CICS environment if the EGL PCB refers to the I/O PCB, a TP PCB, or a GSAM PCB, none of which are available in CICS.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Either modify the system function to reference a valid PCB, or modify the PSB or called parameter list definition to include the referenced PCB

ELA00123P Basic checkpoint used in transaction-oriented BMP

Explanation: A program invoked the `sysLib.commit()` system function while processing as a transaction-oriented IMS BMP. The `sysLib.commit()` system function is implemented as a basic checkpoint (CHKP) function. In the transaction-oriented IMS BMP, this resulted in a read of the message queue that overlaid program storage. The updates to the database have been committed.

This error can only occur if the program uses the `dliLib.AIBTDLI()`, `dliLib.EGLTDLI()`, or `VGLib.VGTDLI()` system functions to read the message queue. The `sysLib.commit()` system function is ignored if the program uses the `get next` statement to read a serial record associated with the input message queue.

The run unit ends.

Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Do not run the program as a transaction-oriented IMS BMP. Alternatively, either remove the use of the `sysLib.commit()` system function or change the `dliLib.AIBTDLI()`, `dliLib.EGLTDLI()`, or `VGLib.VGTDLI()` system function that reads the message queue to a `get next` statement for a serial record and use the resource associations part to associate the serial record with the message queue.

ELA00125P Error number %01D04 is not valid

Explanation: The error handler was called with an error number that it did not recognize. This is a product error.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Ensure that the generated COBOL program has not been modified by generating the program again. Afterwards, run the program again. If the problem persists, do as follows:

1. Record the message number
2. Obtain the dump
3. Record the scenario under which this message occurs
4. Obtain the COBOL source for the problem program
5. Use your electronic link with IBM Service if one is available, or contact the IBM Support Center

ELA00127P A requested function is not supported for form %01C08, FormGroup %02C06

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

The run unit ends.

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

ELA00128P Incompatible attributes for file = %01C08

Explanation: A program is attempting to use a GSAM file that is already opened for another program. The file characteristics (record organization, record length, fixed or variable length records, or key specification) are defined differently for the two programs and the definitions are not compatible.

If the file is EZEPRINT, the problem might be caused by attempting to write forms that do not contain double-byte characters followed by forms that do contain any double-byte data.

The run unit ends.

User response: Define the file characteristics to be the same in both programs or use a different file name for one of the programs.

ELA00129I Form %01C08 was received

Explanation: Related messages give further details.

User response: Refer to the related error messages.

ELA00130P GSAM error, file = %01C08, function = %02C04, status code = %03C02

Explanation: An I/O error occurred on an **add**, **get next**, or **close** statement for a file associated with a GSAM database. Program processing ends on a hard status code if **vgVar.handleHardIOErrors** is set to 0, or on any error status code if there is no **try** block surrounding the I/O statement.

This message can also occur on an implicit OPEN or CLSE call to the GSAM database. An implicit OPEN or CLSE call occurs as a result of an EGL **add** or **get next** statement. Program processing ends on a hard status code if **vgVar.handleHardIOErrors** is set to 0, or on any error status code if there is no **try** block for the **add** or

get next statement that caused the implicit OPEN or CLSE call.

An AI status code for an implicit OPEN might be caused by specifying a file name during EGL resource association that is different from the DD name specified in the GSAM DBD.

For an **add**, message ELA00066I accompanies this message and provides the DL/I I/O area that was used for the call.

The run unit ends. If ELASNAP is allocated, the Rational COBOL Runtime issues a SNAP dump.

User response: Determine the cause of the I/O error from the DL/I status code and either correct the program or the database definition. Refer to the IMS messages and codes manual for your system for an explanation of the DL/I status code.

ELA00131P MSGQ error, file = %01C08, function = %02C04, status code = %03C02

Explanation: An error occurred on a **get next** or **add** statement for a file or a **print** statement for a print form when the file or printer is associated with an IMS message queue (I/O or TP PCB). Program processing ends on a hard status code if **VGVar.handleHardIOErrors** is set to 0, or on any error status code if there is no **try** block surrounding the I/O statement.

Common status codes are:

- QH** Unknown output destination (**add**, **print**, or **converse**)
- A1** Unknown output destination (**add**, **print**, or **converse**)
- A6** Output segment limit exceeded (**add**, **print**, or **converse**)
- FD** Deadlock occurred (**get next**).

For an **add**, **print**, or **converse**, the listed status codes specify that the 8-character system resource name associated with the file or printer at generation or in *recordName.resourceAssociation* or **converseVar.printerAssociation** was not defined to the IMS system as either a terminal or a transaction.

For an **add**, **print**, or **converse** statement, message ELA00066I accompanies this message and shows the DL/I I/O area that was used for the call.

The run unit ends. If ELASNAP is allocated, the Rational COBOL Runtime issues a SNAP dump.

User response: If the output destination is not valid, ensure that it is defined to the IMS system. Also review the program logic to ensure that *recordName.resourceAssociation* or **converseVar.printerAssociation**, if used, are set

correctly. For an explanation of status codes other than the ones listed above, refer to the IMS messages and codes manual for your system.

ELA00132P Variable length %01D02 is not valid for record %02C18

Explanation: The variable length record being written to a GSAM file or a message queue has a length that is greater than the maximum length defined for the record structure. Either the **lengthItem** field contains a value greater than the maximum record length or the **numElementsItem** field contains a value that is greater than the maximum number of occurrences specified.

The first message insert provides the length field that was being used. The length is the total length being written as follows:

- For a GSAM file, the length includes the 2-byte length field itself,
- For a message queue, the length includes the 12-byte header (length, ZZ field, transaction code) itself.

The second message insert provides the name of the serial record being written to the GSAM file or the message queue.

The run unit ends.

Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Modify the program to move a valid value to the **lengthItem** field or to the **numElementsItem** field.

ELA00134P I/O PCB conflict between programs %01C08 and %02C08

Explanation: A program invoked using the **call** or **transfer to program** statement accesses the I/O PCB as a serial file. The initial program in the transaction is a main Text UI program and the current program accesses the I/O PCB. The control logic for a main Text UI program cannot operate correctly when a program that it invokes using the **call** or **transfer to program** statements also accesses the I/O PCB.

The run unit ends.

User response: Modify the called or transferred-to program so it does not access the I/O PCB. Alternatively, call or transfer to the program from a main basic program.

ELA00135P The program is not expecting an input form

Explanation: Another program issued a **show** statement that specified a form, but the receiving program does not specify the **inputForm** property.

The run unit ends.

User response: Either change the invoking program to avoid sending a form or change the receiving program to specify an input form.

ELA00136P DL/I error occurred in work database operation

Explanation: An error occurred during use of the work database when it was implemented using DL/I. This message is accompanied by additional DL/I diagnostic messages, including ELA00061P, that provide additional information about the error. Message ELA00061P includes the DL/I function and status code. Refer to the IMS messages and codes manual for your system for a description of the status code.

The run unit ends.

If ELASNAP is allocated, the Rational COBOL Runtime issues a SNAP dump.

User response: This is a database definition error or an error in the definition of the work database PCB in your IMS PSB. Record this information and any other diagnostic messages, and notify the system administrator.

ELA00137P SQL error occurred in work database operation

Explanation: An error occurred during use of the work database when it was implemented using SQL. This message is accompanied by additional SQL diagnostic messages, including ELA00073P, that provide additional information about the error.

The run unit ends.

If ELASNAP is allocated, the Rational COBOL Runtime issues a SNAP dump.

User response: Determine the cause of the problem from the SQL code and the SQL error information in related message ELA00074I, and correct the database definition.

ELA00138P %01C08 was replaced in the middle of a conversation

Explanation: The program was running in segmented mode and ran a **converse** statement. However, the program was replaced in the load library during user think time (the time between writing the form to the terminal and receiving the user's input).

The program conversation with the user started with the original version of the program and cannot be resumed.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues

a dump based on options selected using the diagnostic controller utility.

User response: Run the program again.

ELA00139P MFS map program %01C06 and MFS map %02C08 have different versions

Explanation: An MFS form services program attempted to process a message input descriptor for an MFS form that was generated at a different time than the MFS form services program. Both the MFS form services program and the form it works with must be built in the same generation step.

This is probably a problem with the installation of either the program or the MFS form after generation of a FormGroup. One of the following might have occurred:

- The MFS form services program might have been compiled and linked without installing the MFS forms, or vice versa.
- The MFS form might have been installed in an MFS test library, but you did not enter an IMS /TEST MFS command prior to starting the transaction.
- The MFS form might have been installed in the MFS production library, and you entered a /TEST MFS command prior to starting the transaction.
- The MFS form might have been used in a **show** statement to transfer from another program. The transfer-from program used a different FormGroup, but the form name on the **show** statement is the same as the **inputForm** name for the transfer-to program.

In the IMS/VS environment, the transaction (logical unit of work) ends and processing continues with the next message. In the IMS BMP environment, the run unit ends.

User response: Ensure that the same version of the MFS form services program and the MFS control blocks are installed in the correct libraries. If the **show** statement and **inputForm** property are involved, ensure that the transfer-from and transfer-to programs use the same FormGroup.

ELA00140P Segmentation storage size discrepancy for %01C08

Explanation: The size of the segmentation storage record is not valid for the specified program.

Possible causes for the error include:

- The program is replaced in the load library in the middle of a program conversation with the user
- The program issues a **show** statement, but the receiving program expects an input form that has different characteristics
- The program is segmented and issues a **converse** statement when **sysVar.transactionID** contains a transaction code, but that transaction code is

associated with a program that has no relationship to the issuing program. If the **sysVar.transactionID** is used to switch transaction codes, the new transaction must start either the same program that was started by the old transaction or the program that issued the **converse** statement.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Try the transaction again. If the program works correctly, the error was caused by a re-link in the middle of the conversation. If the error still occurs, determine why there is a mismatch and correct the situation that caused the error.

ELA00141P Data table %01C08 cannot be modified. Delete %02D06 bytes.

Explanation: The program's attempt to modify a shared DataTable would cause an increase in DataTable size beyond the CICS limit, which is 65535 bytes.

The run unit ends.

User response: Either change the logic of the program so that the DataTable is not modified or decrease the size of the DataTable content by the specified number of bytes.

ELA00142P Form %01C08 in group %02C06 not supported on this device

Explanation: A form has been sent to a device using IMS Message Format Services, but the device type does not correspond to the list of **screenSizes** specified for the form part or the combination of the **mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore** build descriptor options that match the specified screen sizes

- A print form was sent to a destination that is defined as a terminal in the IMS System Generation. The destination is the system resource name specified for EZEPRINT at generation or an override value loaded into the **converseVar.printerAssociation** system variable at run time. The message appears at the terminal where the print form was directed, not at the terminal that originated the transaction. Program processing continues.
- A text form is defined in a FormGroup that contains multiple forms with different values for the **screenSizes** property. The screen size to which the form was directed was not included in the list of **screenSizes** or the combination of the **mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore** build descriptor

options that match the specified screen sizes. The message appears at the terminal that originated the transaction as the result of a **converse** or **show** statement. The program conversation with the user at this terminal ends because there is no way for the user to enter data. The program continues processing with the next message on the message queue.

MFS does not notify the program that a problem has occurred. Therefore, message ELA00142P is built into the MFS source to provide a method of notifying you when an error occurs. A SNAP dump is not issued.

User response: If the error occurred for a print form, review the resource association information specified during generation, the program logic used to set the value of the **converseVar.printerAssociation** system variable and the MFS build descriptor options (**mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore**) to determine the appropriate corrections to make. Depending on the corrections required, generate either the program or FormGroup again. In addition, if the print form was sent to a terminal device, it might be necessary for the system administrator to purge the messages pending for the terminal using the IMS /DEQ command.

If the error occurred for a text form, review the **screenSizes** property specified for this form and the MFS build descriptor options (**mfsDevice**, **mfsExtendedAttr**, and **mfsIgnore**) to determine the appropriate corrections to make. Generate the map group again.

If the program using the text form is a nonconversational program (**spaSize="0"** build descriptor option), the user only needs to clear the screen and type another transaction code to resume work.

If the program that used the terminal map is a conversational program (**spaSize** build descriptor option greater than 0), the user must clear the screen, type /EXIT to end the conversation and then type another transaction code to resume work.

ELA00143P Data table %01C07 is not a message table

Explanation: A message DataTable was specified for the program. The DataTable specified is not a message table.

The run unit ends.

User response: Either define the DataTable as a message table and generate the DataTable again, or correct the **msgTablePrefix** property specified for the program and generate the program again.

ELA00144P Segmentation storage error

Explanation: Segmentation storage has an internal error mapping memory.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: This is an internal system error. Contact the system administrator for assistance.

ELA00145A Form name required - enter /FOR %01C06O form-name

Explanation: The FormGroup has more than one form, but a valid form name was not entered when the IMS /FOR command was used to display the form.

User response: Enter the /FOR command again, using the following format:

```
/FOR FormGroup0 formname
```

ELA00146P Segmentation status error

Explanation: The status byte for segmentation storage management is lost and the program has no way to recover.

This error occurs when a PA key is pressed prior to pressing the ENTER key or a PF key for an IMS conversational transaction.

If the program was generated with a **spaSize** build descriptor option value greater than 0 and without specifying the **spaStatusBytePosition** build descriptor option, then there was no recovery feature generated into the program.

If the program was generated with a **spaSize** build descriptor option greater than 0 and also specified the **spaStatusBytePosition** build descriptor option, then the recovery feature was generated into the program, but was bypassed. A bypass of the recovery feature occurs when a deferred message switch comes from a non-EGL program or an EGL program that was not generated with the same values for the **spaSize**, **spaADF**, and **spaStatusBytePosition** build descriptor options.

In the IMS/VS environment, the transaction (logical unit of work) ends and processing continues with the next message.

User response: Restart the transaction sequence and avoid using PA keys while on an EGL generated screen.

Consider generating the EGL programs with a combination of **spaSize**, **spaADF**, and **spaStatusBytePosition** build descriptor options that

will allow recovery from pressing a PA key.

ELA00147A Key sequence is not valid. Last screen will display - enter the data again

Explanation: A PA key was pressed prior to pressing the ENTER key or a PF key. IMS has reserved the use of the PA keys. All modifications on the previous screen are lost.

User response: Enter the data again and avoid use of PA keys while on an EGL generated screen.

ELA00149I %01C07 command ignored during message database load

Explanation: The PSB for the message database specifies that the database is being initially loaded. Only ADD commands are supported during initial load of a DL/I message database.

User response: Run the message utility again, specifying the PSB for the database.

ELA00151P %01C07 of message record to/from message database failed

Explanation: The message utility program encountered an error inserting or deleting a message in the message database. This message is accompanied by either the DL/I or SQL diagnostic messages describing the error.

If an ELASAP DD statement is specified in the JCL, Rational COBOL Runtime issues a snap dump. The run unit ends.

User response: Review the diagnostic messages. Verify that the database has been successfully defined by checking either the DL/I or the DB2 message database create job (ELAMSJL2) messages. Correct the problem and run the job again.

ELA00152I Message file %01C03 has been added

Explanation: The indicated user message file has been successfully added to the message database.

User response: Test the programs that use this user message file.

ELA00153P %01C08 failed on file %02C08

Explanation: While running the message utility, an attempt was made to access (open, close, read, or write) the indicated file. The access failed and the message utility ended. The first message insert indicates the type of access that failed. The most common errors are a missing DD statement for the file or DCB parameters that are not correct.

User response: Refer to the job listing for system error messages pertaining to the indicated DD name. Correct

the error and run the job again, starting with the command that caused the error.

ELA00154I Message file %01C03 has been replaced

Explanation: The indicated user message file has been successfully replaced in the message database.

User response: Test the programs that use this user message file.

ELA00155I Message file %01C03 has been deleted

Explanation: The indicated user message file has been successfully deleted from the message database.

User response: Change the program using this user message file to use another message file and generate the program again.

ELA00156I Replace on non-existent message file %01C03, file was added

Explanation: A REPLACE command was issued for the indicated message file, but the file did not exist in the message database. The file was added instead.

User response: None, provided the file was added to the correct message database.

ELA00157P %01C08 failed on file %02C08, file status = %03C06

Explanation: While running the message utility, an attempt was made to access (open, close, read, or write) the indicated VSAM file. The file identifies the DD name. The file status consists of the VSAM return code (2 characters), function (1 character), and feedback code (3 characters). The access failed and the message utility terminated. The first message insert indicates that type of access that failed.

User response: Refer to the VSAM administration guide for your system for a definition of the status codes. Also look at the job listing for system error messages pertaining to the indicated DD name. Correct the error and run the job again, starting with the command that caused the error.

ELA00158P Syntax error on command

Explanation: A command being processed by the message utility did not follow the correct syntax. The message utility ends.

User response: Correct the command and run the job again, starting with the command that had the incorrect syntax.

ELA00159P Message file %01C03 already exists in the message database

Explanation: An attempt to add a user message file failed because the message file already existed in the message database for the language specified in the current message utility command. The return code is set to 08.

User response: Use the REPLACE command to update the message file in the message database.

ELA00160P Message file %01C03 does not exist in the message database

Explanation: An attempt to remove or list a user message file failed because the message file does not exist in the message database for the language specified in the current message utility command. The return code is set to 08. If the insert is an asterisk, you attempted to list all messages in an empty message database.

User response: Correct the message file ID in the command and run the job again.

ELA00162P Message I/O error, type %01C04, file %02C08, code %03C08

Explanation: An error occurred when a program generated using Cross System Product/370 Runtime Services Version 1 Release 1 attempted to open or close a user message file. The type variable insert specifies VSAM as the message file type. The file insert specifies the DD name. The first two bytes of the code insert are either 08 (to specify an OPEN) or 16 (to specify a CLOSE). The next two bytes are the ACB (Access control block) return code in hexadecimal format. The remaining bytes in the code insert are zero.

The run unit ends.

User response: Have the administrator do one of the following:

- Determine the cause of the problem from the VSAM error code. First, see Chapter 22, “Common System Error Codes for z/OS Systems,” on page 167 for the tables of common VSAM codes. If the codes are not listed in the tables, refer to the VSAM administration guide for your system for a definition of other VSAM codes. Also verify that the user message file is allocated correctly.
- Convert the message file to a message table and generate the program again under EGL, VisualAge Generator, or CSP/370AD Version 4 Release 1.

ELA00163P %01C08, %02C60

Explanation: This message is used when a Rational COBOL Runtime message cannot be found in the language-dependent message DataTable program ELACxxx, where xxx is the language code.

The first variable insert in this message is the error message number for the error that actually occurred. The second insert in this message contains one of the message inserts that is used by the error that actually occurred. This message is repeated as many times as necessary to report all inserts. The inserts are reported in order by their number: %01, %02, and so on.

User response: See the message with the corresponding message number in this manual. Take the action appropriate for that message. Also, contact the system administrator to determine why the message could not be found in the Rational COBOL Runtime language-dependent message DataTable program.

ELA00164P %01C08, %02C04, %03C02, %04X08

Explanation: The error handler was not successful in using a DL/I call to write diagnostic information about another error to normal destinations for error information. The variable inserts contain the following information:

- Destination from the terminal identifier field of the PCB used in the call.

The destination can be the error destination specified at program generation, the user terminal ID, or the IMS log.

- DL/I function
- DL/I status code
- PCB Address

Rational COBOL Runtime ends the program with a user abend.

User response: For information about locating the diagnostic messages in the dump, see Chapter 19, “Finding Information in Dumps,” on page 151. These messages relate to the original error that ended the program. Also verify that the **errorDestination** value specified in your build descriptor options is included in the IMS system generation.

ELA00166P The recursion stack exceeds the maximum size allowed

Explanation: The stack that contains information to support recursion or segmentation has become too large.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Check for an infinite loop that is causing a large number of recursions. Either limit the

number of recursions, or reduce the number of functions in the program.

ELA00167I The diagnostic message queue is empty

Explanation: The diagnostic print utility for IMS ended without printing any diagnostic messages because the queue was empty.

User response: None required.

ELA00168P %01C03

Explanation: The NLS language code in the file allocated to ELAMSG as shown in the insert is not valid. The Rational COBOL Runtime utility ends because the language code for messages and report headings cannot be determined.

User response: Correct the JCL so that the ELAMSG DD statement references a sequential file or in-stream data that contains a valid NLS code in columns 1 through 3 of the first record. See "Installation and Language-Dependent Options for z/OS" on page 16 for a list of the valid NLS codes.

ELA00169I Work database purged of %01D08 records older than day %02C06, time %03C06

Explanation: The utility that purges obsolete records from the work database has completed normally.

User response: None required.

ELA00170P Input is not valid

Explanation: Either the date or the time provided to the utility that purges obsolete records from the work database was nonnumeric or was not valid.

The run unit ends.

User response: Ensure that the date is in Julian format (YYDDD - two positions for the year and three positions for the day of the year). Ensure that the time is in HHMMSS format (two positions for the hour, two positions for the minutes, and two positions for the seconds). The date and time specified must be at least 24 hours before the time that the purge program is run.

ELA00172I CICS error, system identifier %01C08

Explanation: An error occurred on a CICS function to be performed on a remote system. The message displays the CICS identifier for the remote system.

This message is always issued along with other messages that identify the function being performed and the CICS error return information.

User response: None required.

ELA00173P An error occurred in remote program %01C08, date %02C08, time %03C08

Explanation: An error occurred in a remote program that caused the remote program to stop running. Diagnostic messages might have been logged at the remote location giving information about the error. The date and time stamp on this message can be used to associate the messages logged at the remote system with this error message.

The run unit ends.

User response: Report the error to the system administrator.

ELA00174P %01C08 cannot be used in called programs on a remote system

Explanation: The `sysLib.commit()` and `sysLib.rollback()` system functions cannot be used in a remote called basic program or in a program called by a remote called basic program.

The run unit ends.

User response: Move the `sysLib.commit()` and `sysLib.rollback()` system functions to the program that called the remote program.

ELA00179P An error occurred starting transaction %01C08

Explanation: IMS or CICS indicates that an error occurred when a program attempted to start the specified transaction. A message following this message gives the IMS or CICS error codes.

The run unit ends.

User response: Determine the cause of the error from the following message and correct the error.

ELA00180P Error recovery PCBs not passed to program

Explanation: The program specifies `callInterface = DLICallInterfaceKind.CBLTDLI` and was called by a non-EGL program. Two required PCBs (the I/O PCB and the alternate express PCB) were not passed to the program. The PCBs are required for issuing rollback and commit functions, and reporting error conditions.

The error results in an abend with a dump because the PCBs for reporting and recovering from the error are not available.

The run unit ends.

User response: Modify the program to pass the I/O PCB and the alternate express PCB to the program using one of the following techniques:

- Specify the PCB name as a program parameter and set the `pcbParms` program property.

- Specify **psbData** as a program parameter and set the **psbParm** program property.

ELA00181P I/O PCB not passed to program %01C08

Explanation: The program specifies **callInterface = DLICallInterfaceKind.CBLTDLI** and was called by a non-EGL program. The I/O PCB was not passed to the program. This PCB is required for issuing rollback and commit functions and for reporting error conditions

The run unit ends.

User response: Modify the calling program to pass the I/O PCB to the EGL program. Modify the EGL program to expect the I/O PCB in the parameter list using one of the following techniques:

- Specify the PCB name as a program parameter and set the **pcbParms** program property.
- Specify **psbData** as a program parameter and set the **psbParm** program property.

ELA00183P SYNCPOINT not allowed with PCB parameters

Explanation: The program invoked the **sysLib.commit()** or **sysLib.rollback()** system functions. Each of these functions results in an EXEC CICS SYNCPOINT command, which ends the currently scheduled PSB. Either this program or a program that called this program included a PCB in the called parameter list. The PCB address passed in the parameter list is no longer valid because the PSB is not active.

The run unit ends.

User response: Either modify the program so it does not invoke the **sysLib.commit()** or **sysLib.rollback()** system functions, or modify the program to receive the PSB as a parameter rather than the individual PCBs.

ELA00184P Program %01C08 and form services program %02C08 are not compatible

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

The run unit ends.

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

ELA00185P Length of %01D02 for record %02C18 is not valid and conversion ended

Explanation: Conversion of a variable length record between the workstation format and host format cannot be performed because of one of the following conditions:

- The record length for the current record indicates that the record ends in one of the following:
 - The middle of a numeric field

- The middle of a DBCHAR character
- The middle of an SO/SI string.
- The record is longer than the maximum length defined for the record.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

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

ELA00186P An operand of type MBCHAR in a conversion operation is not valid

Explanation: Conversion of an MBCHAR field from EBCDIC to ASCII or from ASCII to EBCDIC cannot be performed because a double-byte data value is not valid.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Modify the program to ensure that any MBCHAR fields are valid in the records to be converted.

ELA00187P Conversion table %01C08 does not support double-byte character conversion

Explanation: Conversion of an MBCHAR or DBCHAR field from ASCII to EBCDIC or from EBCDIC to ASCII cannot be performed because the specified conversion table does not include conversion tables for double-byte characters.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Modify the program to specify a conversion table that contains the double-byte conversion tables that are valid for DBCHAR and MBCHAR data. For background information, refer to the EGL help topic on data conversion.

ELA00188P Conversion Error. Function: %01C25, Return Code: %02C05, Table: %03C08

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 are:

- The code pages 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 DBCS code page than the code page that is currently in effect on the system.

User response: For background information, refer to the EGL help topic on data conversion.

ELA00191I Program %01C08, generation date %02C08, time %03C08

Explanation: An error in the specified program has occurred. The error is identified in other messages preceding this message. The error might be caused by changes to individually generated components of the program.

User response: Verify the generation date and time of the program with that of other generated components.

ELA00192I Print services program %01C08, generation date %02C08, time %03C08

Explanation: An error in the specified print services program has occurred. The error is identified in other messages preceding this message. The error might be caused by changes to individually generated components of the controlling program.

User response: Verify the generation date and time of the print services program with that of other generated components in the program.

ELA00195I Form group format module %01C08, generation date %02C08, time %03C08

Explanation: An error in the specified FormGroup format module has occurred. The error is identified in other messages preceding this message. The error might be caused by changes to individually generated components of the controlling program.

User response: Verify the generation date and time of the FormGroup format module with that of other generated components in the program.

ELA00201P z/OS %01C08 error in service %02C08, RC = %03D04

Explanation: Rational COBOL Runtime received an error return from a z/OS macro. The inserts identify

the macro name, the Rational COBOL Runtime program name, and the return code.

The run unit ends.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Contact the system administrator.

ELA00202P The file name %01C65 is not valid in the record-specific variable resourceAssociation or in converseVar.printerAssociation

Explanation: The value in either the *recordName.resourceAssociation* or *converseVar.printerAssociation* is not in a valid format. This message can occur when a spool file name has a format that is not valid.

User response: Refer to the EGL help system to determine the valid syntax. Correct and generate the program again.

ELA00203P CICS I/O error on file %01C08, resource %02C08

Explanation: The current program has attempted to gain access to a CICS file, and CICS returned a status code that indicated an I/O error occurred. The file is the logical file name specified in the record part declaration. The resource is the CICS FILE or TDQUEUE resource definition entry.

Possible causes of the error are the following:

- The file does not exist on disk.
- The file is not defined in the CICS FILE or TDQUEUE resource definition entry.
- The file was specified to be opened when first referenced.
- On z/OS CICS, the file was closed using the CSMT or CEMT transactions.
- For z/OS CICS, the DD statement for the file in the CICS startup JCL is missing, does not match the FILE name, or is in error.
- The file has been changed or otherwise corrupted.

Message ELA00204I is also displayed with the information from the EXEC interface block (EIB).

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: Have the system administrator use the CICS diagnostic information in this message and in message ELA00204I to determine the cause of the error. Correct the error and run the program again.

ELA00204I CICS EIBFN %01X04, RCODE %02X12, RESP %03D04, RESP2 %04D04

Explanation: The current program has received an error code for a CICS command.

The run unit ends.

User response: Refer to the CICS application programmers' guide for an explanation of the EXEC interface block (EIB) codes. Correct the error and run the program again.

ELA00205P A CICS %01C22 error occurred in service %02C08

Explanation: Rational COBOL Runtime received an error status code for a CICS command. This message identifies the command and the service program that issued the command. This message is accompanied by message ELA00204I, which contains the response codes from the EXEC interface block (EIB).

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: Have the CICS administrator use the CICS diagnostic information in this message and in message ELA00204I to determine the cause of the error. Correct the error and run the program again.

ELA00206P Format of file %01C08 is not valid, reason code %02C01, resource %03C56

Explanation: The attributes of the system resource associated with the specified file name are not compatible with the properties defined for the record in the program. The reason code identifies the problematic attribute, as follows:

- 1 Key offset
- 2 Key length
- 3 Access method
- 4 Record format
- 5 Record length

An access method mismatch occurs when the type of data set allocated does not match what the program expects. For example, a VSAM file is allocated as a system sequential file or a partitioned data set is allocated as a sequential file without specifying a member name.

The run unit ends.

User response: Change the record part declaration, the resource associations part, or both, so that the record properties match the system resource attributes. Generate and test the affected programs again.

ELA00207P The attributes for file %01C08 are not compatible, reason code %02C01

Explanation: A program has attempted to use a file having file attributes that differ from another program in the run unit. All programs in a run unit must use the same attributes for a file. The reason code identifies the problematic attribute, as follows:

- 1 Key offset
- 2 Key length
- 3 Access method
- 4 Record format
- 5 Record length
- 6 Using the `sysVar.remoteSystemID` system variable to identify the location of a remote file

The run unit ends.

User response: Change the Record part declarations, the resource associations part, or both, so that all programs in the run unit have identical attributes for the file. Generate and test the affected programs again.

ELA00208P Print services program %01C06 and FormGroup format module %02C08 were generated separately

Explanation: The specified print services program attempted to process a form that was generated at a time different from the FormGroup format module. Both the print services program and the FormGroup format module must be generated at the same time.

The run unit ends.

User response: Make sure that the print services program and the FormGroup format module were generated at the same time and are installed in the correct libraries.

ELA00209P The cursor position is (%01D02, %02D02). That position is outside of the current form: %03C08.

Explanation: The cursor position is outside of the form boundaries because of the `ConverseLib.setCursorPosition` function, which sets the position of the cursor for the next converse.

User response: Change the function so that the cursor position is within the form boundaries.

ELA00209I Backout completed successfully after abnormal termination for transaction %01C04

Explanation: The specified CICS transaction ended abnormally with the code specified in accompanying

message ELA00222P. Rational COBOL Runtime termination was successful in backing out all changes to recoverable resources and closing all open external resources associated with the transaction.

User response: No action required.

ELA00210P Service number %01D04 is not valid

Explanation: An attempt was made to start a Rational COBOL Runtime routine that does not exist or that is not valid.

The run unit ends.

In CICS environments, Rational COBOL Runtime issues a dump based on options selected using the diagnostic controller utility.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Generate and test the program again.

If the problem persists, do as follows:

1. Record the message number
2. Obtain the dump
3. Record the scenario under which this message occurs
4. Obtain the COBOL source for the problem program
5. Use your electronic link with IBM Service if one is available, or contact the IBM Support Center

ELA00212P Error encountered gaining access to file %01C08, spool resource %02C65

Explanation: An error was received when attempting to gain access to a spool file. The message is accompanied by message ELA00204I, which contains response codes from the CICS EXEC interface block (EIB).

If the function was a write spool request (EIBFN 5602) and the spool resource name was specified as node ID without being qualified by user ID, an error will occur if the user did not log on using the CICS logon procedure.

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: If the spool resource name specifies node ID without specifying user ID, log on using the CICS logon procedure before running the program again. Otherwise, refer to the CICS customization documentation for an explanation of the codes that are returned by the spool interface; then, correct the problem specified in the response codes.

Refer to the EGL help system for additional information on the format of the system resource name.

ELA00215P PSB does not match Enterprise Generation Language PSB definition

Explanation: The number of PCBs passed to the program at program initialization time was less than the number of PCBs in the EGL PSB record definition. This message is accompanied by ELA00217I.

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: Do as follows:

- Correct the DL/I PSB; or
- Correct the EGL PSB record definition and generate the program again

ELA00216P CICS DL/I error, function %01C04, UIBFCTR %02X02, UIBDLTR %03X02

Explanation: CICS detected an error in a DL/I call. The message variable inserts specify the function being requested and the return codes from the CICS user interface block (UIB). If the function code is PCB, the program was attempting to schedule the program PSB. The message is accompanied by message ELA00217I.

Common return codes are as follows:

UIBFCTR	UIBDLTR	Description
08	00	Argument on DL/I call not valid. This error can occur if the IMSESA installation option in module ELARPIOP is specified as YES, but the IMS environment is not IMS/ESA.
08	01	PSB not found. The PSB must be defined to CICS.
08	03	The calling program has already successfully issued a scheduling (PCB) call that has not been followed by a TERM call.
08	05	PSB initialization was not successful.
08	06	The PSB in the scheduling call is not defined in the program control table (DLZACT).
08	07	A TERM call was issued when the task had already been terminated.

UIBFCTR	UIBDLTR	Description
08	09	An MPS batch program attempted to issue a PCB call for a read-only PSB or for a nonexclusive PSB if program isolation was active.
08	FF	DL/I not active
0C	02	Intent scheduling conflict

The run unit ends.

User response: If the DL/I call is not valid, check the definition of the call to the `dlLib.AIBTDLI()`, `dlLib.EGLTDLI()`, or `vgLib.VGTDLI()` system function. Otherwise, correct the problem specified by the error code. For additional codes, refer to the CICS application programmers' guide for your system to determine the meaning of the error codes.

ELA00217I Program %01C08, PSB name %02C08

Explanation: An error was detected in the specified DL/I program. The message is accompanied by messages ELA00215P or ELA00216P, which identify the problem.

The run unit ends.

User response: Refer to the accompanying messages for the problem cause.

ELA00218P Invocation of sysLib.audit not successful, journal id = %01D05, journal type = %02C02

Explanation: This message is accompanied by ELA00204I, which displays the contents of EIBRESP.

Common EIBRESP codes for CICS are as follows:

- | | | |
|----|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 22 | LENGERR | The computed length for the journal record exceeds the total buffer space allocated for the journal data set as specified in the journal control table (JCT) entry for the data set |
| 43 | JIDERR | Occurs if the specified journal identifier does not exist in the JCT |

The run unit ends.

User response: Refer to the CICS resource definition guide to define journal data sets, or contact the system administrator.

ELA00219P %01C22 error for %02C06 file %03C08, %04C56

Explanation: An I/O operation was not successful for the specified file.

Program processing ends on any nonzero status code if the I/O statement is not in a **try** block; and ends on a hard error if the I/O statement is in a **try** block when `vgVar.handleHardIOErrors` is set to 0.

The message identifies the I/O statement, the file type, the file name as specified in the record part, and the system resource name associated with the file.

The run unit ends.

In all z/OS environments, Rational COBOL Runtime issues a SNAP dump if the ELASNAP data set is allocated.

User response: Check that the correct data set has been allocated for this file.

ELA00220P Dynamic allocation was not successful, file %01C08, return %02D04, error reason code %03X04.

Explanation: Rational COBOL Runtime was not successful in an attempt to perform dynamic allocation for the specified file. The other inserts are the return code in register 15 and the error reason code returned by the SVC 99 instruction.

The most common cause is that the file was not available. If you want your program to receive control after getting the I/O error value `fileNotAvailable`, place the I/O statement in a **try** block and set `vgVar.handleHardIOErrors` to 1. If either condition is not met, Rational COBOL Runtime ends the program.

The run unit ends.

User response: Contact the system administrator. See the *z/OS MVS Authorized Assembler Services Guide* for an explanation of the codes.

ELA00221P File %01C08, system resource name %02C56, not found

Explanation: Rational COBOL Runtime attempted to dynamically allocate the file with the system resource name shown in the message. The file could not be found.

If the system resource name is a 1- to 8-character DD name, then there is no DD card for the file in the job JCL. If the system resource name is a data set name, then the data set either does not exist or is not cataloged.

The run unit ends.

User response: If the name is a DD name, allocate a file to the DD name in the JCL. If the name is a data set name, ensure that the file exists and is cataloged.

ELA00222P Transaction %01C04 ended abnormally with CICS abend code %02C04

Explanation: The specified CICS transaction ended abnormally with the specified code.

Message ELA00021I is displayed after the main message, showing the function name and statement line number where the abend occurred. The function name is accurate only if you set the symbolic parameter WRITEFUNCTIONDETAILS to YES in the generated EGL COBOL program. Specifying YES for this symbolic parameter causes additional COBOL statements to be generated to ensure that the function name is continuously updated with the correct value. The statement line number is accurate only if you set the symbolic parameter WRITESTATEMENTDETAILS to YES in the generated EGL COBOL program. Specifying YES for this symbolic parameter causes additional COBOL statements to be generated to ensure that the statement line number is continuously updated with the correct value.

On z/OS CICS systems, the following additional information is provided:

- On CICS Version 2 systems, if the ABEND code is ASRA or ASRB, this message is accompanied by the message ELA00223P and the ABEND exit can determine the module within which the error occurred.
- On later CICS systems, if the abend code is ASRA or ASRB, CICS message DFHAP0001 identifies the offset in the module at which the error occurred. The diagnostic control option specified for transaction abends using the Rational COBOL Runtime diagnostic control utility determines whether a dump occurs.

The Rational COBOL Runtime abend handler ends the program by issuing another ABEND command using the same code.

User response: See Chapter 23, “Rational COBOL Runtime Return Codes, Abend Codes, and Exception Codes,” on page 185 for a description of abend codes using the format ELAx. See “Common CICS Abend Codes” on page 195 for CICS or user program documentation for an explanation of other abend codes.

ELA00223P Program %01C08 abended at offset %02X08

Explanation: The specified program has abended with an ASRA or ASRB abend code. This indicates that a program check has occurred at the specified hexadecimal offset.

Rational COBOL Runtime ends the program with a user abend.

Message ELA00021I is displayed after the main message, showing the function name and statement line number where the abend occurred. The function name

is accurate only if you set the symbolic parameter WRITEFUNCTIONDETAILS to YES in the generated EGL COBOL program. Specifying YES for this symbolic parameter causes additional COBOL statements to be generated to ensure that the function name is continuously updated with the correct value. The statement line number is accurate only if you set the symbolic parameter WRITESTATEMENTDETAILS to YES in the generated EGL COBOL program. Specifying YES for this symbolic parameter causes additional COBOL statements to be generated to ensure that the statement line number is continuously updated with the correct value.

User response: If the program is a generated COBOL program, use the compile listing to find the COBOL verb that was running when the program ended abnormally. The COBOL comments identify the EGL statements associated with the COBOL verb. Determine from the dump whether the problem was caused by bad data passed to the program. If the generated COBOL program is in error, use your electronic link with IBM Service or contact the IBM Support Center.

ELA00225P Temporary storage queue name %01C08 is not valid

Explanation: The record-specific variable *recordName.resourceAssociation* is set to a temporary storage queue name that is not valid. The name conflicts with a queue name that is reserved for Rational COBOL Runtime. Names cannot begin with EZE.

The run unit ends.

User response: Specify a valid temporary storage queue name in the program.

ELA00228P The program attempted to use the resource %01C65 with file %02C07 and file %03C07

Explanation: The program attempted to associate the same system resource with two different files. The resource cannot be associated with two different files at the same time.

The run unit ends.

User response: Examine the program and correct the logic. Generate and test the affected programs again.

ELA00229P Invocation of sysVar.startTransaction failed, transID = %01C04, terminal ID = %02C08

Explanation: This message is accompanied by the message ELA00204I, which displays the contents of EIBRESP.

Common codes are as follows:

11 TERMID error

The specified terminal ID is not known to CICS.

28 TRANSID error

The specified transaction ID is not known to CICS.

The run unit ends.

User response: Have the system administrator define the terminal or transaction to CICS.

ELA00230P An error was encountered accessing CICS queue %01C08

Explanation: An error was received when attempting to access a CICS queue. The queue can be a transient data queue or temporary storage queue. This message is accompanied by message ELA00204I, which contains response codes from the CICS EXEC interface block (EIB).

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: Refer to the CICS application programmers' guide for an explanation of the response codes.

ELA00231P Error encountered retrieving data passed to program %01C08

Explanation: An error was received when attempting to retrieve data being passed to this program by a **transfer to transaction** or **show** statement or by a **vgLib.startTransaction()** system function. This message is accompanied by message ELA00204I, which contains response codes from the CICS EXEC interface block (EIB).

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: Refer to the CICS application programmers' guide for an explanation of the codes that are returned.

ELA00232P Form %01C08 in FormGroup %02C06 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.

The run unit ends.

User response: Specify the correct **screenSizes** property for the form. Generate the FormGroup again.

If you are running on a CICS system, have the system

administrator check that the alternate screen size for your device type is specified in the PCT entry for your transaction.

If the FormGroup name uses the format ELAxxx, where xxx is the language code, the FormGroup might have been modified incorrectly. The ELAxxx FormGroup contains the Rational COBOL Runtime error forms.

ELA00237P CICS TS Queue %01X16 error occurred in work database operation for program %02C08

Explanation: An error was received when attempting to access a CICS temporary storage queue. This message is accompanied by message ELA00204I, which contains response codes from the CICS EXEC interface block (EIB).

If the error is an INVREQ (EIBRESP=16), the problem might be caused by Rational COBOL Runtime attempting to write a record that is longer than the control interval size for the VSAM data sets used for the auxiliary storage queue. The maximum segmentation record size written by Rational COBOL Runtime is set by the TSQUE option in the installation options module ELARPIOP. TSQUE specifies the maximum size as the number of kilobytes; the default value is 16 KB.

The run unit ends.

User response: Refer to the CICS application programmers' guide for an explanation of the codes.

If the control interval size is the problem, have the system administrator assemble the installation module again after setting the TSQUE value to a value less than the control interval size.

Refer to the Rational COBOL Runtime program directory for your system for more information.

ELA00239P Print services program %01C08 cannot support print request from program %02C08

Explanation: A program and a print services program were generated with different values for the **formServicePgmType** build descriptor option. The print services program does not contain the type of print support (GSAM or SEQ) required by the program.

The run unit ends.

User response: Generate the FormGroup again with the **formServicePgmType** build descriptor option required by the program. Be sure to include all the types of printing that are required for any program that uses the FormGroup.

ELA00249P Mapping services program %01C08 compiled with DATA(31) cannot be used by program

Explanation: A form services program compiled with the DATA(31) compiler option has been loaded for a program link-edited as AMODE(24).

User response: Compile the form services program again with the COBOL DATA(24) option; and make sure that the **data** build descriptor option is set to 24 whenever the FormGroup is generated.

ELA00250P Program cannot process data with 31-bit addresses

Explanation: The initial program in the run unit was compiled with DATA(31). The current program was link-edited as AMODE(24). This is not compatible.

User response: Do one of the following:

- Compile the initial program in the run unit as DATA(24).
- Link-edit the current program as AMODE(31).

ELA00251P Data table %01C08 compiled with DATA(31) cannot be used by program

Explanation: A DataTable compiled with the DATA(31) compiler option has been loaded for a program link-edited as AMODE(24).

User response: Compile the DataTable program again with the COBOL DATA(24) option. Also ensure the **data** build descriptor option is set to 24 whenever the DataTable is generated.

ELA00252P Error on file %01C08, queue name %02C08, RC = %03C08

Explanation: An I/O logic error was detected by Rational COBOL Runtime during processing of an I/O statement for a CICS temporary storage queue.

Program processing ends on any nonzero status code if the I/O statement is not in a **try** block; and ends on a hard error if the I/O statement is in a **try** block when **vgVar.handleHardIOErrors** is set to 0.

Because the error was detected by Rational COBOL Runtime instead of the access method, the return code value consists of the characters RS (for runtime services) followed by a Rational COBOL return code number.

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

User response: See Chapter 22, "Common System Error Codes for z/OS Systems," on page 167 to determine the meaning of the Rational COBOL return

code, and take the appropriate action.

ELA00253P Program %01C08 was not generated to receive form %02C08

Explanation: The specified program received a form as an input form, but the program does not contain processing logic for handling segmented programs. Either the wrong transaction name was specified when the program was started, or the wrong program was specified in the transaction definition.

The program was started as a result of one of the following:

- Specifying a new value for the **sysVar.transactionID** system variable before issuing a **converse** statement in a segmented Text UI program instead of using the original transaction code. After the user entered input data, processing returned to the wrong program because the new transaction code is not associated with the program that issued the **converse** statement.
- In IMS/VS, using the /FORMAT command for a form that specifies the transaction code for the program.

The program must specify either an **inputForm** property or have the **segmented** property set to YES and issue a **converse** statement for the form being received.

The run unit ends.

User response: Make sure that the following are specified correctly:

- The transaction ID specified on the **show** statement
- The form name in the **inputForm** program property
- The transaction ID contained in **sysVar.transactionID** system variable before a segmented **converse**

Generate the modified program again.

ELA00254P Invalid values for sysLib.audit, journal ID = %01D05, type = %02C02, length = %03D05

Explanation: A parameter in **sysLib.audit()** is not valid:

- The journal ID must be between 1 and 99
- The third byte in the record must be in the range X'A0' to X'FF'
- The record length must be between 28 and 32763

The run unit ends.

Rational COBOL Runtime issues a dump based on information supplied for the transaction with the diagnostic controller utility.

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

ELA00255P Invalid values for sysLib.audit, type = %01C02, length = %02D05

Explanation: A parameter in `sysLib.audit()` is not valid:

- The third byte in the record must be in the range 'XA0' to 'XFF'
- The record length must be between 28 and 32767

The run unit ends.

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

ELA00260E %01D08 bytes of VGUI record do not fit in %02D08 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.

ELA00261E sysLib.setError message information and inserts do not fit in %01D08 byte buffer

Explanation: The program invoked the `sysLib.setError()` system function one or more times to write messages associated with a VGUI record. The information associated with the last message written does not fit into the buffer used by the program for communicating with the user.

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

ELA00262E VGWebTransaction program and VGUI record bean %01C18 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: Ensure 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.

ELA00263E Number of elements value %01C10 is out of range for structured field array at offset %02X08

Explanation: A VGWebTransaction program could not write a VGUI record because the value in the `numElementsItem` field for a structured field array in the record was less than 0 or greater than the maximum size 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.

ELA00264E Input data entered by the user does not fit in the 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 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.

ELA00265E Segmented converse is not supported when local variables or function parameters are in the run-time stack

Explanation: The message indicates that a **converse** statement is not valid because the EGL run time cannot restore the values of function parameters or local variables after the **converse** runs.

For more information, refer to the EGL help topic on segmentation.

The runtime stack is a list of functions; specifically, the current function plus the series of functions whose running made possible the running of the current function.

User response: Modify the program in one of two ways:

- Ensure that the functions on the runtime stack have neither parameters nor local variables
 - Ensure that the **converse** is not segmented.
-

ELA00266E MQ function %01C08, Completion Code %02C02, Reason Code %03C08.

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

- 1 MQCC_WARNING
- 2 MQCC_FAILED

The reason for the completion code is set in the reason code field by MQSeries®. Some common reason codes are:

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

The run unit ends.

User response: Refer to the *MQSeries Application Programming Reference* for further information on MQSeries completion and reason codes.

ELA00267E Queue Manager Name %01C48.

Explanation: This is the name of the queue manager associated with the failing MQ function call listed in message ELA00266. If the failing MQ function was MQOPEN, MQCLOSE, MQGET, or MQPUT, the name identifies the queue manager 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.

The run unit ends.

User response: Refer to the *MQSeries Application Programming Reference* for further information on the MQSeries completion and reason codes that are listed in message ELA00266.

ELA00268E Queue Name %01C48.

Explanation: This is the name of the queue object associated with the failing MQ function call listed in message ELA00266.

The run unit ends.

User response: Refer to the *MQSeries Application Programming Reference* for further information on

MQSeries completion and reason codes that are listed in message ELA00266.

ELA00269E Array index value %01D07 out of range for array %02C18 with size of %03D07

Explanation: The index specified for the dynamic array is out of bounds.

User response: Specify an index between 1 and the current number of elements in the array.

ELA00270E An attempt was made to exceed the maximum size of array %01C18

Explanation: An attempt was made to add an element to a dynamic array that already contains the maximum allowed number of elements.

User response: Modify the program in either of two ways:

- Increase the value of the dynamic array property **maxSize**
- Change the logic so that the number of elements is always less than or equal to the value of **maxSize**.

ELA00300I A new copy was requested for part %01C08

Explanation: A new copy was requested for the programs associated with the specified part. Newly started transactions use the new copy of the program.

User response: None required.

ELA00301I The diagnostic control options were changed

Explanation: The diagnostic control options were changed after a user request from the Rational COBOL Runtime Diagnostic Control utility.

User response: None required.

ELA00302I Error message queue sent to print destination

Explanation: The contents of the transient data queue containing the error messages were sent to the spooling system after a user request from the Rational COBOL Runtime Diagnostic Print utility.

User response: None required.

ELA00303I Error message queue sent to print destination and deleted

Explanation: The contents of the transient data queue containing the error messages were sent to the spooling system after a user request from the Rational COBOL Runtime Diagnostic Print utility. The contents of the transient data queue were then deleted.

User response: None required.

ELA00304A Type a valid selection number, then press Enter

Explanation: The selection number entered for a field on one of the Rational COBOL Runtime utility panels is not valid. The cursor is positioned at the field in error.

User response: Type a valid selection and press Enter.

ELA00305A Type a name, then press Enter

Explanation: A required field was left blank on one of the Rational COBOL Runtime utility panels. The cursor is positioned at the empty field.

User response: Type a valid name and press Enter.

ELA00306P CICS new copy was not successful for program %01C08. Press F2.

Explanation: The CICS SET NEWCOPY command was not successful for the specified part. The specified part was requested on the Rational COBOL Runtime New Copy panel.

User response: Press F2 to view message ELA00204I, which contains the CICS response information from the EXEC interface block (EIB). Verify that the part name is correct. Refer to the CICS application programmers' guide for an explanation of the EXEC interface block (EIB) codes.

ELA00308P I/O error on error message queue. Press F2.

Explanation: A CICS error occurred when attempting to gain access to the error destination queue identified on the Rational COBOL Runtime Diagnostic Print panel.

User response: Press F2 to view message ELA00204I, which contains the CICS response information from the EXEC interface block (EIB). Verify that the error destination name is correct. Refer to the CICS application programmers' guide for an explanation of the EXEC interface block (EIB) codes.

ELA00309A Error message queue was not found

Explanation: The error destination queue identified on the Rational COBOL Runtime Diagnostic Print panel was not found.

User response: Specify the correct error destination queue name on the panel.

ELA00310A Type a valid response, then press Enter.

Explanation: A value that was not recognized was specified in the field where the cursor is positioned. Valid values are shown following the field on the form.

User response: Type a valid value in the field and press Enter.

ELA00313I Default options are in effect for this transaction

Explanation: You made a request to view the diagnostic control options in effect for a specific transaction. The options currently in effect for the transaction are the default options.

User response: To exit, press F3. To change the options for this transaction do as follows:

1. Type the new options
2. Select action 1
3. Press Enter

ELA00314I Error message queue was empty

Explanation: A request was made to print an error message queue that does not contain any messages.

User response: None required.

ELA00315I Trace transaction list was updated successfully

Explanation: The list of transactions you specified to be traced has been processed successfully.

User response: None required.

ELA00316I Trace filter criteria updated successfully

Explanation: The list of trace filter criteria you specified has been processed successfully.

User response: None required.

ELA00317P Service number is not valid

Explanation: The trace filter criteria contains a service number that is not valid. For z/OS batch or IMS BMP, if this error is detected during ELATRACE data set parsing, the run unit ends.

User response: Do one of the following:

- For z/OS batch or IMS BMP, correct the service number specification in the ELATRACE data set and run the program again.
- For CICS or IMS/VS programs, correct the service number.

ELA00318P Tag in %01C08 is not valid

Explanation: The filter criteria contains a tag that is not valid. Valid tags are FILTER, EFILTER, APPLS, EAPPLS, SERVICES, and ESERVICES.

The run unit ends.

User response: Correct the tag specification and run the program again.

ELA00319P Missing or misplaced tag in %01C08

Explanation: The filter criteria contains a missing or misplaced tag.

The run unit ends.

User response: Correct the filter criteria and run the program again.

ELA00320P Too many programs in %01C08

Explanation: The filter criteria contains too many programs. The maximum number is 16.

The run unit ends.

User response: Reduce the number of programs or remove all program filter criteria, then run the program again.

ELA00321P Too many services in %01C08

Explanation: The filter criteria contains too many services. The maximum number is 32.

The run unit ends.

User response: Reduce the number of services or remove all service filter criteria, then run the program again.

ELA00322P One or more filters has a invalid value

Explanation: One or more codes entered for the DATASTREAM, TRACETOFILE, APPSTMT, SQLIO, SQLERR or IDUMP filters is not valid. The valid code that is entered must be either Y (yes) or N (no).

For z/OS batch or IMS BMP, the run unit ends.

If you are defining filters online for z/OS CICS or IMS/VS, the filter containing the value that is not correct is highlighted.

User response: Do one of the following:

- For z/OS batch or IMS BMP, specify either Y or N for these filters and run the program again.
 - For CICS or IMS/VS, type one of the valid values for the highlighted filter as shown on the form, then press Enter.
-

ELA00323P I/O error on storage queue %01C08. Press F2.

Explanation: An error was received when attempting to access a temporary storage queue in the diagnostic message print utility. Press F2 to view message ELA00204I, which contains response codes from the CICS EXEC interface block (EIB).

User response: Refer to the CICS application programmers' guide for an explanation of the codes.

ELA00324P Error reading trace control record. Press F2.

Explanation: An error was encountered when attempting to read or write to the trace control record in CICS. Press F2 to view more information.

For z/OS CICS, message ELA00204I is displayed, which contains response codes from the CICS EXEC interface block (EIB).

User response: Review the accompanying error messages.

ELA00325P Error opening %01C08

Explanation: An error was encountered when attempting to open the specified data set.

User response: Make sure that the data set has the correct attributes.

ELA00326P Error reading %01C08

Explanation: An error was encountered when attempting to read the specified data set.

User response: Make sure that the data set has the correct attributes.

ELA00342A The maximum number of copies already exists for the DataTable

Explanation: The maximum number of copies of a DataTable that can be used in a CICS region at one time is 5. The request for a new copy of the DataTable was rejected.

User response: Old copies of a DataTable that are in use are freed when all the transactions that are using the DataTable end. Retry the new copy request later.

ELA00363P An incompatible terminal configuration change has been detected

Explanation: Rational COBOL Runtime detected a change to a terminal that is different from the previous terminal on which the program was running. Changing terminal configurations while a program is running is not supported.

The run unit ends.

User response: Restart the program.

ELA00364I Snap dump is in progress

Explanation: This is an informational message which is displayed on the screen to inform you that a problem has occurred and that a snap dump is being taken.

User response: The snap dump could take a while. When the snap dump is complete, a Rational COBOL Runtime error panel is generally displayed with messages indicating what went wrong.

ELA03001I F3=EXIT F8=CONTINUE

Explanation: None.

User response: None required.

ELA03002I F3=EXIT

Explanation: None.

User response: None required.

ELA03003I CLEAR=EXIT

Explanation: None.

User response: None required.

ELA03004I PF3=EXIT PF8=FORWARD

Explanation: None.

User response: None required.

ELA03005I PF3=EXIT

Explanation: None.

User response: None required.

ELA03006I PA1=CONTINUE

Explanation: None.

User response: None required.

ELA03007I IBM Rational COBOL Runtime

Explanation: None.

User response: None required.

ELA09937I Function name %01C48

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

User response: None required.

ELA09938P An error occurred when trying to invoke a service function.

Explanation: Rational COBOL Runtime was unable to transfer control to the specified service function.

User response: Make sure that the service is available to your program and you specified the correct function name.

ELA09939P Service binding must be a web binding.

Explanation: You specified a service binding, but when Rational COBOL Runtime tried to invoke the binding, it was not a web binding.

User response: Correct the binding and regenerate.

ELA09940I Binding Key: %01C75.

Explanation: This message provides the service binding key for which a problem occurred. Other related messages provide the information about the actual cause of the error.

User response: None required.

ELA09941P An error occurred when trying to invoke a Web Service function, JNI setup error %01D06.

Explanation: A problem occurred transferring control to a service function in the iSeries® environment.

User response: Verify that you meet the requirements in the "Special considerations for generating EGL or web services in iSeries environments" help topic.

ELA09942I Service property name %01C48

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.

ELA09943E Required service property does not exist in service module %01C08

Explanation: The required service property does not exist in the service module. Message ELA09942I provides the name of the service property that was required.

User response: Make sure you are using the correct service property name.

ELA09944I Entry point name %01C48

Explanation: This message provides the name of the entry point 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.

ELA09945E Cannot find entry point in service module %01C08

Explanation: The requested entry point does not exist in the service module Message ELA09944I provides the name of the entry point that was requested.

User response: Make sure you are using the correct entry point name.

ELA09946E Reference target cannot be resolved in service module %01C08

Explanation: The reference target does not exist in the service module Message ELA09948I provides the name of the reference target that was requested.

User response: Make sure you are using the correct reference target name.

ELA09947E Component reference missing target in service module %01C08

Explanation: The component reference does not exist in the service module Message ELA09949I provides the name of the component reference that was requested.

User response: Make sure you are using the correct component reference name.

ELA09948I Reference name %01C48

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: None required.

ELA09949I Component name %01C48

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: None required.

ELA09951I Service target name %01C48

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.

ELA09952E Cannot find service target in service module %01C08

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: Make sure you are using the correct service target name.

ELA09954E 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.

ELA09955E 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.

ELA09956E 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.

ELA09958E 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.

ELA09959E 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.

ELA09960E 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.

ELA09961E 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.

ELA09962E 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.

ELA09963E DL/I exception

Explanation: A DL/I 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.

ELA09964E 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.

ELA09965E 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.

ELA09966E 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.

ELA09967E Exceeded max size on array exception

Explanation: A dynamic array exceeded its maximum specified size.

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.

ELA09968E Append arrays of mismatched size exception

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

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

ELA09969E Insufficient heap memory exception

Explanation: The program ran out of memory.

User response: Try to resolve the problem using one of the following methods:

- For z/OS batch, increase the REGION parameter in the runtime JCL.
- For any COBOL environment, set the HEAPSIZ symbolic parameter to 16384 and generate the first program in the run unit again. Note that HEAPSIZ 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 HEAPSIZ 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 HEAPSIZ resolves the problem, contact IBM support to determine if you need to apply maintenance for your EGL server product.

ELA09970E Attempting to access an uninitialized dynamic array exception

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

User response: Change the program logic to ensure that the dynamic array is initialized. You can initialize the dynamic array by using the **new** operator or a set value block at declaration time.

ELA09971E Invalid format used in format function call exception

Explanation: The program invoked one of the formatting functions with an invalid format mask. The functions for which this error can occur include: **strLib.formatDate()**, **strLib.formatTime()**, **strLib.formatTimeStamp()**, and **strLib.formatNumber()**. The mask can be specified in several ways including the following:

- As the format argument for the system function
- In a system variable

For example, for **strLib.formatDate()**, you can specify the date format mask by including the optional second

argument for the system function or by setting the **strLib.defaultDateFormat** system variable.

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

ELA09972E 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.

ELA09973I Condition code %01C04

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

ELA09974E Unhandled exception occurred.

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

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

ELA09973I Condition code %01C04

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.

ELA09974E Unhandled exception occurred. EGL line: %01C06

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

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

FZE messages

**FZE10014I ABEND %01C04 HAS OCCURRED,
TRAN= %02C04 %03C08 %04D05**

Explanation: CICS has detected an abend in the specified transaction. The time and date the abend was detected is listed. This message appears in the CSMT queue. If the abend is an ATNI, then the following information will also appear:

```
DATASTREAM FROM LAST TD QUEUE RECORD READ:  
data in hex format... *data in character format*  
DATASTREAM SENT TO THE DEVICE:  
data in hex format... *data in character format*
```

The above information shows the last transient data queue record read, as well as the data sent to the device which caused the ATNI abend. The data appears in both hex and character format, much like the data would appear in a CICS transaction dump.

User response: The Rational COBOL Runtime print transaction continues to run. Determine the cause of the CICS abend and run the transaction again if desired.

**FZE10040I PRINT TRANSACTION NOT
STARTED FROM TRANSIENT DATA**

Explanation: The Rational COBOL Runtime print transaction (program FZETPRT) received control for other than a transient data queue trigger level. Probable cause: XSPP entered at a terminal.

User response: Contact your system administrator.

FZE10060P PARAMETER ERROR

Explanation: One or more of the input parameters was specified incorrectly.

User response: If you were initializing a file, check the parameter list you specified. Correct it and try the procedure again. This message should not occur during the installation procedure. If this error occurs during installation, contact the IBM Support Center for assistance.

**FZE10061P ERROR OPENING %01C08 REG
15=%02X03, ERR=%03X03**

Explanation: An error occurred while attempting to open the named VSAM file.

User response: Look up the return code in register 15 and the feedback (or reason) code in the appropriate VSAM manual for your operating environment. Correct the problem and try this procedure again.

**FZE10062P ERROR WRITING %01C08 REG
15=%02X03, ERR=%03X03**

Explanation: An error occurred while attempting to

write to the specified VSAM file.

User response: Look up the return code in register 15 and the feedback (or reason) code in the appropriate VSAM manual for your operating environment. Correct the problem and try this procedure again.

FZE10064I SUCCESSFUL COMPLETION

Explanation: This step in the installation procedure FZEZVCPO finished correctly.

User response: None required.

FZE10065I RECORDS READ: %01D08

Explanation: This shows the number of records read from the source statement library or from the system logical unit SYSIPT.

User response: None required

FZE10066I RECORDS WRITTEN: %01D08

Explanation: The indicated number of records were written to the VSAM output file.

User response: None required.

FZE10067I FILE %01C08 ALREADY LOADED

Explanation: The specified output file has already been loaded or initialized. This message occurs when a file is being initialized or conditionally loaded.

User response: None required.

**FZE10068P SOURCE LIB I/O ERROR FOR FILE
%01C08**

Explanation: There was an error reading from the specified input file.

User response: Check the listings for the return codes from the previous steps of the installation procedure to determine if the source statement library installed correctly. If the return code was not zero, correct the problem and run the previous step again. Then run this step again.

FZE10069P MISSING SOURCE MEMBER %01C08

Explanation: The specified source library member necessary for input to this step in the installation procedure is missing.

User response: Check the listings for the return codes from the previous steps of the installation procedure to determine if the source statement library installed correctly. If the return code was not zero, correct the

problem and run the previous step again. Then run this step again.

PRM messages

PRM00001P Invalid parameter group name %01C08

Explanation: The parameter group name specified is not valid. Parameter group names may be 1 through 8 alphanumeric characters.

User response: Correct the parameter group name and retry the request.

PRM00002I New parameter group being defined

Explanation: You have entered a parameter group name which has not been previously defined. You may enter the parameters for the new parameter group to complete this definition. If you do not enter any parameters and you press Enter to save the group, then an empty group will be created.

User response: None required.

PRM00003P Invalid selection character

Explanation: You have entered a selection code which is not valid. Valid selection codes are:

- 'S' Select a parameter group for update.
- 'D' Delete an existing parameter group.

User response: Correct the selection character and retry the request.

PRM00004P Already at top or bottom of list

Explanation: You attempted to do one of two things:

- Scroll forward on the last screen of the list
- Scroll backward on the first screen of the list.

No scrolling occurred.

User response: Do not attempt to scroll beyond the start or the end of the list.

PRM00005I Function key not supported

Explanation: You have used a function key that is not supported by the facility. The keys which are available are described in the top portion of the form.

User response: Check the description of what functions are available, and use a different function key.

PRM00006I Specified parameter group(s) not found

Explanation: You have requested to view a list of parameter groups, and no parameter group exists for the search conditions you have specified.

If you entered a question mark (?) to view a list of all

parameter groups, then your parameter group file is empty.

User response: If you have made an error, then correct the problem and retry the request.

PRM00007P Unexpected I/O error occurred, RC = %01C08

Explanation: You have attempted an operation against the parameter group file and an I/O error has occurred. The operation was not completed.

This error indicates some damage has occurred to the parameter group file. This error should be corrected before any further maintenance to your parameter groups is attempted.

User response: Contact your Systems Programmer.

PRM00008P File is full, parameter group cannot be added

Explanation: You have attempted to add a parameter group to your parameter group file, which is full. The parameter group has not been added.

User response: Review your existing parameter groups to determine if any of them can be deleted. Deleting existing parameter groups will make room for new groups that you want to add. If you are not able to delete any existing parameter groups, then the parameter group file must be redefined to allow more entries.

PRM00009I Operation(s) successfully completed

Explanation: You have successfully completed the operation requested. The possible operations are:

- Addition of a new parameter group.
- Modification of an existing parameter group.
- Deletion of an existing parameter group.

User response: None required.

PRM00010P Parameter group file EZEPRMG not found

Explanation: Either the name was specified incorrectly or the file is not properly defined to the system.

User response: Ensure the parameter group file is defined and associated with EZEPRMG as the FILE entry name on CICS systems.

PRM00011P Unable to connect to parameter group file EZEPRMG

Explanation: The Parameter Group Utility was unable to connect to the parameter group file. The file must be associated and defined to the system.

User response: Verify the file name specified has been defined and associated with EZEPRMG as the FILE entry name on CICS systems.

Notices

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

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

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

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

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

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
3600 Steeles Avenue East
Markham, ON
Canada L3R 9Z7

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

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE: This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

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. 2000, 2011. All rights reserved.

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

This IBM Rational COBOL Runtime Guide for zSeries documents the intended Programming Interfaces that allow the customer to write programs to obtain the services of Rational Business Developer.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.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 <http://www.ibm.com/legal/copytrade.html>.

The following terms are trademarks of other companies:

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

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

Index

Special characters

- /FORMAT command 111
- /HOLD command 111
- /MODIFY command 110
- /WORKDBType build descriptor option
 - IMS 13

A

- abend
 - ASPE
 - CICS 39
 - codes
 - CICS 185, 195
 - COBOL 193
 - IMS runtime 194
 - non-CICS environments 187
 - preparation 163
 - system 191
 - dumps
 - COBOL 151
 - Rational COBOL Runtime 151
 - recovery considerations
 - z/OS 45, 48, 49, 57, 58
- activating trace sessions
 - CICS 156
- adding
 - file name to the CICS file control table
 - z/OS 42
 - job control statements
 - z/OS 41
- addressing, extended 31
- alternate index, defining 28
- alternate PCB, using 52
- American National Standards printer control character
 - z/OS 33, 35
- AMODE 6, 7
- analyzing
 - detected errors 148
- application
 - load module storage for Rational COBOL Runtime 5
 - plan for DB2 14
- applying maintenance to
 - Rational COBOL Runtime 3
- ASA (see also American National Standards printer control character) 35
- ASPE abend, preventing 39
- attributes for DBCS, hardware 31

B

- backing up data 32
- backup, maintaining copies of production libraries 118
- batch
 - print services program 79

BIND

- command
 - data set 72
 - default 78
 - defining 78
- DB2 programs 32
- precompile messages 164
- buffer size, printing
 - CICS 37
- build descriptor
 - and compiler options that affect performance 27
 - options
 - commentLevel 149
 - errorDestination 140, 145, 147
 - imsFastPath 140, 193
 - imsLogID 140, 146
 - language code 145, 146, 148
 - mfsDevice 193, 194
 - mfsExtendedAttr 193, 194
 - mfsIgnore 193, 194
 - mfsUseTestLibrary 193
 - performance considerations 27
 - restoreCurrentMsgOnError 140
 - spaSize 53, 108
 - targetNLS 192
 - trace 155
- output files 74

C

- catastrophic error 143
- cautions
 - empty KSDS data set, VSAM restriction 30
 - PRTMPP parameter, line skip malfunction 37
- CEDA transaction, RDO 91
- change or view
 - defaults - ELAC04 128
 - options - ELAC02 126
- checking
 - access authorization
 - z/OS 32
 - database authorization
 - CICS 32
 - IMS 57
- CICS
 - abend codes 195
 - activating trace sessions 156
 - database
 - recovery considerations 45
 - DB2 considerations 10, 45
 - destination control table (DCT)
 - printing, DBCS 38
 - sample entry 44
 - transient data queue name 35
 - diagnostic control options 126
 - DL/I considerations 9, 45
 - ELAC transaction 126
 - ELAM transaction 121

CICS (continued)

- ELAN transaction 122
- ELAU transaction 124
- EZEEZ transaction 35
- file descriptions 33
- installation considerations 9
- mode, pseudoconversational, residency consideration 40
- monitoring and tuning 10
- new modules 94
- parameter group
 - print file 33
- parameter group, creating and maintaining 129
- PCT (program control table), printing, DBCS 38
- performance
 - considerations 39
- preparation 91
- PRIN transaction 35
- print destination, specifying in DCT 44
- printing
 - buffer size 37
 - DBCS (double-byte character set) 35, 38
 - DCT (destination control table) 35, 44
 - destination control table (DCT) 35, 44
 - double-byte character set (DBCS) 35
 - EZEEZ transaction 35
 - file description 33
 - form-feed 35
 - FORMFD=NO parameter 35
 - FZETPRT program 36, 44
 - parameter, PRITYP 38
 - PCT (program control table), FZETPRT program 38
 - PRIN transaction 35
 - printer destination 44
 - program control table (PCT) 38, 44
 - PRTBUF parameter 36
 - PRTMPP parameter 36
 - PRITYP parameter 36
 - SEND command 37
 - terminal control table (TCT), entry 44
 - transient data queue 44
- processing mode
 - types 34
- program control table (PCT)
 - DTB=YES and DBP value 44
 - printing, DBCS 38
- pseudoconversational
 - processing mode 40
 - programs and residency 40
- residency
 - considerations 39, 40
 - general rules 40

- DB2 (*continued*)
 - precompile
 - messages 164
 - program plan 14
 - programs
 - bind 32
 - work database
 - clearing records 59
 - expanding the table space 61
 - IMS 13
 - multiple 63
- DBCS (double-byte character set)
 - data on a non-DBCS terminal 112
 - hardware attributes 31
 - printing
 - CICS 35, 38, 44
- DBRMLIB 72
- DCAPRMG file, parameter group for FZETPRT 36
- DCT (destination control table)
 - entries 92
 - printing, DBCS 38
 - sample entry 44
 - transient data queue 35
 - trigger level 35
- DD statements by file type 98
- deactivating a trace session 161
- default
 - print destination, IMS 53
- defining
 - alternate index 28
 - data files 41
 - ESDS (serial) data set 28
 - KSDS (indexed) data set 28
 - program data files 41
 - program specification block (PSB)
 - IMS 52
 - RRDS (relative) data set 28
 - transient data
 - extrapartition 44
 - intrapartition 44
 - transient data files
 - extrapartition 44
 - intrapartition 44
 - transient data queues
 - extrapartition 43, 44
 - intrapartition 43, 44
 - VSAM data files 28
- deleting old records from the work database 58
- descriptions
 - CICS files 33
 - IMS files 51
- destination control table (DCT)
 - entries 92
 - printing, DBCS 38
 - sample entry 44
 - transient data queue 35
 - trigger level 35
- destination, default print, IMS 53
- detecting errors 139
- determining position in program 153
- DFHAC2016 messages 195
- DFHAC2206 messages 195
- DFS057I error message 193
- DFS064 error message 193
- DFS182 error message 193

- DFS2082 error message 113, 193
- DFS2766I error message 113, 194
- DFS555I error message 112, 193
- diagnosing problems 139
- diagnostic control
 - facility
 - CICS utilities 121
 - options
 - change or view defaults 128
 - change or view options 126
 - ELAC transaction 126
- diagnostic message print utility, ELAU 124
- disk storage requirements
 - for Rational COBOL Runtime 8
- DL/I
 - considerations
 - CICS 9, 45
 - IMS 58
 - z/OS batch 9, 48
 - integrity and recovery considerations
 - CICS 45
 - IMS 58
 - z/OS batch 49
 - status codes 180
 - work database
 - clearing records 59
 - expanding the database 60
 - in IMS 13
 - multiple 63
- double-byte character set (DBCS)
 - hardware attributes 31
 - printer 44
- DSNX100I messages 164
- dumps
 - snap, listing file on IMS 51
- dynamic
 - interface plan 32
 - storage utilization in Rational COBOL Runtime 7

E

- ELA2SSQL module 55
- ELA2SSQX module 55
- ELA2SSQY module 55
- ELAC, diagnostic control options 126
- ELAC02 panel, change or view options 126
- ELAC04 panel, change or view defaults 128
- ELACJWKD member 63
- ELADIAG file 51
- ELAM, CICS utilities menu 121
- ELAN, new copy utility 122
- ELANCC module 55
- ELAPCB macro 52
- ELAPRINT system output file 47, 51
- ELARPRTM load module 55
- ELARPRTR load module 55
- ELARSDCB load module 55
- ELASNAP file 51
- ELAU, diagnostic message printing utility 124
- ELAWKJC member 59
- ELAWKJCD member 59
- ELAWORK work database PCB 52

- ELAWORK2 DL/I work database 63
- emulating IBM 3270 devices 31
- error
 - detection 139
 - message
 - file 51
 - panel 144
 - reporting 139
 - IMS 140
 - in IMS 112
 - summary 141
- errorDestination message queue 145
- ESDS (serial) define cluster 28
- expanding
 - the table space (DB2) 61
 - work database 60
- express alternate PCB 52
- extended addressing considerations
 - z/OS 31
- external work file, backing up 32
- extrapartition transient data, defining 44
- EZEBIND data set 72
- EZEDESTP special function word 47
- EZEJCLX data set 72
- EZEPCT data set 73
- EZEPPT data set 73
- EZEPRINT data set
 - IMS 53
 - specify as PRO1 44
- EZEPRMG file
 - CICS 33
 - parameter group for FZETPRT 36
- EZESRC data set 73
- EZETRACE data set 47
- EZEZ transaction 35, 44

F

- FCT (file control table)
 - entries 93
 - user data file 42
- file
 - control table (FCT)
 - described 93
 - default message queue, IMS 51
 - description
 - CICS 33
 - IMS 51
 - descriptions 33
 - error message 51
 - from generation 74
 - parameter group 33
 - snap dump listing, IMS 51
 - system output 47
 - trace 47
- file control table (FCT)
 - entries 93
 - user data file 42
- form feed
 - order (see American National Standards printer control character) 35
 - printing 35
- FORMFD parameter
 - option=NO, forms alignment 35
 - parameter group for PRIN or EZEZ 38

FORMFD parameter (*continued*)
 used with FZETPRT program 36

FormGroup
 format module 79

function
 new copy 39
 preload, IMS 54

FZETPRT program 38
 DBCS considerations 38
 PRIN or EZEZ transaction 44
 special parameter group 36
 terminal printing support in CICS 35

FZEZREBO utility, initializing indexed files 30

G

generated applications
 with PL/I programs 16

generating
 application control block 52

H

hardware attributes for DBCS 31

I

IBM 3270 device, emulating 31

IBM 5550 family of terminals 31

IDCAMS program
 BLDINDEX command 28
 DEFINE PATH command 28
 loading indexed files 31
 REPRO command 28, 30

IGYOP3091W error message 164

IGYOP3093W error message 165

IGYOP3094W error message 165

IGYPA3013W error message 165

IGYPG3113W error message 165

IGYPS2015I error message 164

IGYPS2023I error message 164

IGYSC2025W error message 165

improving
 performance 56
 library lookaside (LLA) 28
 link pack area (LPA) 28
 virtual lookaside facility (VLF) 28
 response time 56

IMS
 control region 13
 database
 authorization checking 57
 integrity considerations 54
 recovery considerations, DB2 57
 recovery considerations, DL/I 54
 DB2 considerations 12
 default
 message queue file 51
 print destination 53
 DL/I considerations 58
 ELAPCB macro 52
 error
 controlling, generation
 options 140
 messages 112

IMS (*continued*)
 error (*continued*)
 reporting 112

file descriptions 51

HIPERSPACE buffer usage 56

installation considerations 12

integrity considerations, DB2 57

log format 146

logical unit of work 58

monitoring and tuning 13, 57

new modules 110

performance considerations 56

preload function 54

preloading
 program modules 56
 Rational COBOL Runtime
 modules 55

preparation 107

processing modes 53

program specification block,
 defining 52

residency considerations 54

runtime
 abend codes 194
 messages 193

security considerations 12

segmented mode 53

snap dump listing file 51

system considerations 51

system definition
 batch program as an MPP 108
 batch-oriented BMP program 109
 general 13
 interactive program 107
 parameters 107
 transaction-oriented BMP 109

system printing considerations 53

work database considerations
 DB2 13
 DL/I 13

IMS DC monitor facilities 13

IMS/ESA exploitation 12

IMS/VS, message format service (MFS)
 Control Blocks 56

IMSPARS 57

indexed (KSDS) data set
 define cluster 28
 loading 30

installation considerations
 preparing to install 3

integrity considerations, database
 DB2
 CICS 45
 IMS 57
 DL/I
 CICS 45
 IMS 58
 z/OS batch 49
 IMS 54

intrapartition transient data
 defining 44

J

JCL
 by environment 97

JCL (*continued*)
 examples of runtime 102, 103, 104,
 114, 115
 modifying 97, 98
 modifying runtime 98
 tailoring before generation 97
 templates 97

job stream data set
 runtime 72

K

KSDS (indexed) define cluster 28

L

LE
 runtime messages 192

library
 backup 32
 production copies, maintaining
 backup 118

link pack area
 loading 55
 performance considerations 55

listing file
 IMS, snap dump 51

load library data set 73

load module
 preloading 55
 storage for Rational COBOL
 Runtime 5
 storage for Rational COBOL Runtime
 application 5

loading
 modules into link pack area 55

logical unit of work (LUW)
 IMS 57, 58

M

macro, ELAPCB 52

maintaining
 backup copies of production
 libraries 118
 work database 58

maintenance, applying to
 Rational COBOL Runtime 3

message
 format services
 considerations 64
 description 31, 64
 queue file, default, IMS 51

message format service (MFS) control
 blocks in IMS 56

messages
 DFHAC2016 195
 DFHAC2206 195
 DFS057I 193
 DFS064 193
 DFS182 193
 DFS2082 113, 193
 DFS2766I 113, 194
 DFS555I 112, 193
 DSNX100I 164
 IGYOP3091W 164

- messages (*continued*)
 - IGYOP3093W 165
 - IGYOP3094W 165
 - IGYPA3013W 165
 - IGYPG3113W 165
 - IGYPS2015I 164
 - IGYPS2023I 164
 - IGYSC2025W 165
 - IMS runtime 193
 - preparation 163
 - runtime
 - IMS 193
 - z/OS 195
 - z/OS runtime 195
- MFS
 - control blocks 109
- mode
 - CICS execution, performance considerations 40
 - processing
 - CICS 34
 - IMS 53
- models
 - JCL 97
- modifying
 - IMS system definition parameters 107
 - JCL or CLISTS 97
 - runtime
 - JCL 98
- modules
 - CICS 94
 - IMS 110
 - in memory 28
 - loading into link pack area 55
 - preloading 56
- monitoring and tuning
 - CICS 10
 - IMS system 13, 57
 - performance 57
- moving prepared programs
 - z/OS 117
- multiple work databases 63

N

- new copy
 - function 39
- new copy utility 122
- new copy utility, ELAN 122
- new modules
 - CICS 94
 - IMS 110
- nonsegmented processing mode,
 - CICS 34

O

- object library data set 73
- objects generated
 - application COBOL program 77
 - batch print services program 79
 - BIND command 78
 - FormGroup format module 79
 - from generation 74
 - online print services program 79

- objects generated (*continued*)
 - runtime
 - JCL 77
 - table program 78
 - online print services program 79
 - option
 - preloading
 - program modules, IMS 56
 - Rational COBOL Runtime modules, IMS 55
 - recovery 39
 - SPA 53
 - output of program generation 74

P

- panels
 - Parameter Group Definition (PRGM02) 132
 - Parameter Group Specification (PRGM00) 131
- panels, Parameter Group List Display (PRGM01) 132
- parameter
 - group associated with FZETPRT program
 - DCAPRMG file 36
 - EZEPRMG file 36
 - resident 40
 - WORK in ELAPCB 52
- Parameter Group Definition panel (PRGM02) 132
- parameter group file, EZEPRMG data set, CICS 33
- Parameter Group List Display panel (PRGM01) 132
- Parameter Group Specification panel (PRGM00) 131
- passing transient data between CICS transactions 44
- PCT (program control table)
 - entries 92
 - FZETPRT program 38
- performance
 - considerations 27
 - CICS 39
 - general 15, 28
 - IMS 54, 56
 - IMS/ESA 56
 - z/OS batch 49
 - generation and compiler options 27
 - HIPERSPACE buffers for IMS 56
 - library lookaside (LLA) 28
 - limiting MFS control blocks 56
 - link pack area 28
 - monitoring and tuning
 - IMS 13, 57
 - preload modules 110
 - RES(YES) parameter, RDO DEFINE PROGRAM command 92
 - tuning IMS 57
 - virtual lookaside facility (VLF) 28
- Performance Analysis and Reporting System (PARS) 57
- PL/I programs 16
- plan, DB2 32

- PPT (processing program table)
 - entries 91
- PR01 transient data queue 44
- precompile messages
 - BIND 164
 - DB2 164
- preloading
 - objects, IMS 54
 - print services
 - description 110
 - module 55
 - program 56
 - program 110
 - program modules 55, 56
 - Rational COBOL Runtime modules, IMS 55
 - service module 55
 - table modules 55, 110
- preparation
 - abend codes 163
 - messages 163
- preparing
 - and running programs
 - CICS 91
 - IMS 107
 - z/OS batch 101
 - to install Rational COBOL Runtime 3
- PRGM00 (Parameter Group List Display panel) 132
- PRGM00 (Parameter Group Specification panel) 131
- PRGM02 (Parameter Group Definition panel) 132
- PRIN transaction 33, 35, 44
- print destination
 - CICS, specifying in DCT 44
 - default
 - IMS 53
- print file, utilities 33
- print services program
 - object of generation 79
 - preloading 56
- printing
 - buffer size 37
 - CICS
 - considerations 33
 - file descriptions 33
 - CICS, destination control table (DCT) 35
 - considerations
 - IMS 53
 - DBCS (double-byte character set), printer 44
 - DCT (destination control table)
 - transient data queue name 35
 - trigger level 35
 - default, print destination 35
 - destination control table (DCT)
 - transient data queue name 35
 - trigger level 35
 - destination, using
 - sysLib.startTransaction() system function 35
- diagnostic information
 - CICS 147
 - IMS 145
- EZEZ transaction 35

- printing (*continued*)
 - file descriptions, CICS 33
 - form-feed 35
 - FORMFD=NO parameter 35, 38
 - FZETPRT program 36
 - parameter
 - FORMFD 36, 38
 - group associated with FZETPRT program 36
 - PRTBUF 36
 - PRTMPP 36, 37
 - PRTTYP 36, 38
 - PCT (program control table), FZETPRT program 38
 - PR01 transient data queue 44
 - PRIN transaction 35
 - print destination, default 35
 - printer destination 44
 - program control table (PCT), FZETPRT program 38
 - SEND command 37
 - sysLib.startTransaction() system function for print destination 35
 - transient data
 - at a terminal device 44
 - transient data queue 35, 44
- problem
 - diagnosis 139
- processing
 - batch 47
- processing mode
 - CICS
 - types 34
 - IMS 53
- processing program table (PPT)
 - entries 91
- production libraries, maintaining copies
 - for backup 118
- profile block
 - program 152
- program
 - bind DB2 32
 - data files, defining 28
 - entries 91
 - module, preloading 55
 - preloading 56
 - profile block 152
 - return codes 185
- program communication block (PCB)
 - alternate 52
 - ELAPCB macro 52
- program control table (PCT)
 - DTB=YES and DBP value 44
 - entries 92
 - FZETPRT program 38
- program specification block (PSB)
 - defining 52
 - generation 52
- PRTBUF parameter
 - specifying print buffer size 36
 - using with the FZETPRT program 36
- PRTMPP parameter
 - specifying maximum print positions 37
 - using with FZETPRT program 37
- PRTTYP parameter
 - DBCS printing 38

- PRTTYP parameter (*continued*)
 - using with the FZETPRT program 36
- pseudoconversational
 - processing mode
 - CICS 34, 40

R

- Rational COBOL Runtime
 - abend dumps 151
 - application load module storage 5
 - applying maintenance 3
 - COBOL dynamic storage 6
 - COBOL external storage for non-CICS environments 6
 - control block 152
 - control options by transaction 126
 - customizing JCL procedures 16
 - DB2 considerations
 - CICS 10
 - IMS 12
 - IMS work database 13
 - z/OS batch 9
 - default control options 128
 - diagnostic control options 126
 - disk storage requirements 8
 - DL/I considerations
 - CICS 9
 - IMS work database 13
 - z/OS batch 9
 - dynamic storage 7
 - error 144
 - extended addressing 31
 - generated programs
 - using with PL/I programs 16
 - IMS/ESA exploitation 12
 - installation considerations
 - CICS 9
 - IMS 12
 - preparing to install 3
 - load module
 - reentrant 5
 - storage 5
 - storage estimates, statically linked 6
 - new copy 122
 - performance considerations 15
 - security considerations
 - all systems 15
 - CICS 10
 - IMS 12
 - storage facilities for CICS, using 7
 - storage requirements 5
 - temporary storage queues 12
 - utilities
 - diagnostic control facility (ELAM) 121
 - diagnostic control options (ELAC) 126
 - diagnostic message printing utility (ELAU) 124
 - for CICS 10
 - new copy (ELAN) 122
 - virtual storage requirements 5
 - work database space for segmented applications 8

- Rational COBOL Runtime (*continued*)
 - WSCLEAR option for COBOL, specifying 16
- Rational COBOL Runtime, utilities, parameter group utility, PRGM 131
- Rational COBOL Runtime, utilities, PRGM, parameter group utility 131
- RCT 93
- RDO (resource definition online), generation output 76
- RDO CEDA transaction 91
- recovery
 - options
 - specifying 39
- recovery considerations
 - DB2
 - CICS 45
 - IMS 57
 - DL/I
 - CICS 45
 - IMS 58
 - z/OS batch 49
 - IMS 54
- reentrant code 28
- reentrant load module storage estimates
 - for Rational COBOL Runtime 5
- relative (RRDS) define cluster 28
- reporting
 - errors 139
 - problems 161
- request module, DB2 72
- residency
 - considerations
 - CICS 39
 - IMS 54
 - general rules, CICS 40
- resident
 - parameter 40
 - programs 95
- resource
 - control table 93
 - tables for CICS 91
- Resource Measurement Facility II 57
- response time, improving 56
- return codes
 - Rational COBOL Runtime 171
 - SQL 178
 - system 167
 - sysVar.errorCode 167, 170
- RMF 57
- RRDS, data set definition 28
- running
 - main programs under z/OS
 - batch 101
 - programs under IMS 111
- running under
 - CICS 95
 - IMS
 - BMP with DB2 115
 - main batch as BMP 114
 - main program under BMP 113
 - z/OS batch
 - main batch with DL/I 103
 - main batch with no database 102
 - main batch with no DB2 102
- runtime
 - JCL 77, 98

runtime (*continued*)
 job stream data set 72
 messages
 IMS 193
 z/OS 195
 messages, LE 192

S

sample JCL
 BMP with DB2 115
 IMS BMP program 114
 z/OS batch with DB2 Access 102
 z/OS batch with DB2 and DL/I 104
 z/OS batch with DL/I Access 103
 z/OS batch without DB2 102

saving storage space 55

security considerations
 CICS 10
 general 15
 IMS 12

segmented processing mode
 CICS 34
 IMS 53

SEND command, printing 37

serial (ESDS) define cluster 28

service module, preloading 55

services, message format 31

sharing modules 55

snap dump listing file, IMS 51

spaSize build descriptor option 53, 108

spool files, CICS 11

SQL
 considerations 32
 return codes 178

starting
 IMS programs
 /FORMAT command
 (transaction) 111
 directly (main) 111
 MPPs (transactions) 111

startup JCL for CICS 94

statistics, performance 57

status 13
 codes
 DL/I 180
 key values, COBOL 182

storage requirements
 for Rational COBOL Runtime COBOL
 dynamic storage 6

subsystem ABEND dumps 151

support for DBCS terminals 31

sysLib.startTransaction() system function,
 print destination 35

SYSLIN 73

SYSLMOD 73

SYSOUT system output file 47

SYSPRINT system output file 47

system
 abend codes 191
 considerations
 CICS 33
 general 27
 IMS 51
 definition, IMS 13
 output file 47
 return codes 167

SYSUDUMP system output file 47

sysVar.errorCode 167
 compatibility considerations 167
 return codes 170

T

table
 modules, preloading 55
 preloading 56
 program 78
 space
 expanding 61, 62
 requirements 61

TCT (terminal control table) 44

templates
 CLIST 97
 JCL 97

temporary storage queues 12

terminal control table (TCT) 44

terminal printing
 CICS 35

trace facility 155

trace file 47

tracing
 activating 156
 deactivating 161

transaction
 entries 92

transient data
 defining extrapartition 44
 printing 44
 queue
 defining 43
 printing, CICS 35
 TYPE=INTRA entry in DCT 35

tuning
 IMS 13, 57

U

unit of work, logical
 IMS 57, 58

user data set 72

using
 data build descriptor option 11
 generated applications with PL/I
 programs 16
 multiple work databases 63
 remote files, CICS 43
 using spool files 11

utilities
 diagnostic control options
 (ELAC) 121, 126
 diagnostic message printing
 (ELAU) 124
 for CICS with Rational COBOL
 Runtime 10
 IMS diagnostic message print 135
 new copy (ELAN) 122

utilities, diagnostic, message print utility,
 CICS 124

utilities, parameter group utility,
 PRGM 131

V

virtual storage
 considerations and residency 40
 requirements
 Rational COBOL Runtime 5

VSAM
 data set definition 28
 defining an alternate index 28
 file loading 30
 indexed (KSDS) data set 28
 relative (RRDS) data set 28
 serial (ESDS) data set 28
 status codes 181

W

warnings
 empty KSDS data set, VSAM
 restriction 30
 PRTMPP parameter, line skip
 malfunction 37

work database
 clearing records 58
 deleting old records 58
 ELAPCB macro 52
 expanding 60
 IMS 13
 maintaining 58
 multiple 63
 space for segmented applications 8

WORK parameter in ELAPCB 52

WSCLEAR option for COBOL 16

Z

z/OS
 DB2 considerations for Rational
 COBOL Runtime 9
 DL/I considerations 9
 DL/I considerations for Rational
 COBOL Runtime 9
 installation considerations 3
 preparation 101
 runtime messages 195

z/OS batch
 DL/I considerations 48

z/OS/XA considerations 31



Product Number: 5655-R29

Printed in USA

SC31-6951-06

