



BEA eLink Java Adapter for Mainframe

WLS Edition User Guide

BEA eLink Java Adapter for Mainframe 4.0
WLS Edition
Document Edition 1.0
April 2000

Copyright

Copyright © 2000 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, WebLogic Enterprise, WebLogic Commerce Server, and WebLogic Personalization Server are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA eLink Java Adapter for Mainframe WLS Edition User Guide

Document Edition	Part Number	Date	Software Version
1.0	N/A	April 2000	BEA eLink Java Adapter for Mainframe 4.0 WLS Edition

Contents

About This Guide

Who Should Read This Guide	ix
System Administrators	x
Java Application Developers	x
CICS Application Developers	x
IMS Application Developers	xi
How this Guide Is Organized	xi
Product Documentation	xii
How to Use The Documentation	xiii
Document Conventions	xiii
Contact Us	xv

1. Introducing eLink Java Adapter for Mainframe WLS Edition

JAM Environment	1-1
WebLogic Server	1-2
System Network Architecture Communications Resource Manager	1-3
Java Communications Resource Manager Gateway	1-3
EgenCobol Code Generator	1-4
Supported Third-Party SNA Stack	1-5
Configuration Options	1-5
Combined Configuration	1-6
Distributed Configuration	1-7
Installation Considerations	1-8

2. Configuring the Java Adapter for Mainframe Environment

Configuring the JAM Environment	2-2
Step 1 - Install a Supported SNA Stack	2-2

Step 2 - Install the Java JDK	2-3
Step 3 - Install WebLogic Server	2-3
Step 4 - Install the Java Access for Mainframe	2-5
Step 5 - Install the SNA Communications Resource Manager	2-5
Step 6 - Establish Your Mainframe Environment	2-6
Step 7 - Start WebLogic Server.....	2-6
Step 8 - Configure the Java Communication Resource Manager Gateway (JCRMGW)	2-6
JC_REMOTE_DOMAINS Section.....	2-7
JC_SNACRM Section.....	2-8
JC_SNASTACKS Section	2-9
JC_SNALINKS Section.....	2-10
JC_LOCAL_SERVICES Section	2-12
JC_REMOTE_SERVICES Section	2-13
Sample Configuration File	2-15

3. Developing Java Applications

Building the Base Java Application.....	3-1
Data View Concepts.....	3-2
Obtaining the COBOL Copybook	3-5
Creating a New Copybook	3-5
Using an Existing Copybook.....	3-6
Script Comments	3-7
Script Comments	3-10
Generating the Java Application Source.....	3-10
Generating a Servlet-Only JAM Application	3-11
Creating a Script.....	3-11
Processing a Script	3-15
Generated Files.....	3-16
Customizing a Servlet-Only JAM Application	3-17
Generating a Client Enterprise Java Bean-Based Application	3-26
Creating a Script.....	3-26
Processing the Script	3-29
Generated Files.....	3-29
Customizing an Enterprise Java Bean-Based Application.....	3-34

Generating a Server Enterprise Java Bean-Based Application	3-41
Creating a Script.....	3-42
Processing the Script.....	3-43
Generated Files	3-43
Customizing a Server Enterprise Java Bean-Based Application	3-47
Generating a Stand-Alone Client Application	3-51
Processing a Script	3-53
Generated Files	3-54
Customizing a Stand-Alone Java Application	3-55
 4. Deploying Applications	
Deploying Servlets	4-2
Deploying Enterprise Java Beans	4-3
 5. Security	
 6. Programming Reference	
Field Name Mapping Rules.....	6-2
Field Type Mappings.....	6-2
Group Field Accessors	6-3
Elementary Field Accessors	6-4
Array Field Accessors	6-5
Fields with REDEFINES Clauses	6-5
COBOL Data Types	6-6
 7. Programming Scenarios	
Scenario A: Developing a Multi-Service Data Entry Servlet.....	7-2
Task 1: Use EgenCobol to Create a Base Application.....	7-2
Prerequisite.....	7-2
Step 1: Prepare EgenCobol Script.....	7-3
Step 2: Add Service Entries	7-3
Step 3: Add Page Declaration in EgenCobol Script	7-3
Step 4: Add Servlet Name.....	7-4
Step 5: Generate the Java Source Code	7-4
Step 6: Review the Java Source Code.....	7-5
Task 2: Create Your Custom Application from the Base Application.....	7-6

Step 1: Start with Imports.....	7-6
Step 2: Declare the New Custom Class.....	7-7
Step 3: Add Implementation for doGetSetup.....	7-7
Step 4: Continue Implementation for doGetSetup	7-7
Step 5: Finish Implementation for doGetSetup.....	7-8
Step 6: Create Implementation for doPostSetup	7-8
Step 7: Continue Implementation for doPostSetup	7-9
Step 8: Continue Implementation of doPostSetup	7-9
Step 9: Continue Implementation for doPostSetup	7-10
Step 10: Finish Implementation of doPostSetup.....	7-10
Step 11: Create Implementation for doPostFinal	7-11
Step 12: Update the jermgw.cfg File with Service Entries	7-11
Step 13: Create Basic Three-Part HTML Frame.....	7-12
Step 14: Create a Series of Links to HELP Pages.....	7-12
Task 3: Update the JAM Configurations and Update WLS Properties....	7-13
Task 4: Deploy Your Application	7-14
Task 5: Use the Application	7-15
Scenario A Summary.....	7-17
Create	7-18
Read.....	7-19
Update	7-20
Delete	7-21
Scenario B: Enhancing an Existing Servlet to Originate a	
Mainframe Request	7-22
Task 1: Use EgenCobol to Create a Base Class	7-22
Prerequisite.....	7-22
Step 1: Prepare EgenCobol Script.....	7-22
Step 2: Generate the Java Source Code.....	7-23
Step 3: Review the Java Source Code.....	7-23
Task 2: Update the Survey Servlet Using the Generated Class.....	7-24
Step 1: Start with Imports.....	7-24
Step 2: Add New Data Members.....	7-25
Step 3: Update doPost with Mainframe Request	7-25
Step 4: Continue Updating doPost by Extracting Form Data	7-26
Step 5: Continue Updating doPost by Calling Mainframe Service...	7-26

Task 3: Update the JAM Configurations and Update WLS Properties....	7-27
Task 4: Deploy Your Application	7-28
Task 5: Use the Application	7-29
Scenario B Summary.....	7-30
Scenario C: Updating an Existing EJB to Service a Mainframe Request	7-31
Task 1: Use EgenCobol to Create a Base Class	7-31
Prerequisite.....	7-31
Step 1: Prepare EgenCobol Script.....	7-31
Step 2: Generate the Java Source Code	7-32
Step 3: Review the Java Source Code.....	7-32
Task 2: Update the Trader Interface Using the Generated Class	7-33
Step 1: Start with Import.....	7-33
Step 2: Continue with Imports	7-34
Step 3: Update EJB with dispatch.....	7-34
Step 4: Continue Updating EJB with dispatch.....	7-35
Step 5: Finish Updating EJB with dispatch	7-36
Task 3: Update the JAM Configurations.....	7-36
Task 4: Deploy Your Application	7-36
Task 5: Use the Application	7-37
Scenario C Summary.....	7-38

A. Code Generator Reference Pages

EgenCobol	A-1
Synopsis	A-1
Script Syntax Reserved Words.....	A-2
General rules	A-3
Grammar.....	A-4
Results of Running the Code Generator.....	A-6

B. Configuration Checker Utility

bea.sna.jcrmgw.jcrmConfigurator	B-2
Synopsis.....	B-2
Description	B-2
GWBOOT	B-3
Synopsis.....	B-3
Description	B-3

C. Error and Informational Messages

D. Java Docs

Glossary

Index

About This Guide

This guide provides information about the BEA eLink Java Adapter for Mainframe (JAM) WLS Edition, a product that enables client/server transactions between Java applications and OS/390 CICS or IMS programs.

This section covers the following topics:

- Who Should Read This Guide
- How this Guide Is Organized
- Product Documentation
- Contact Us

Who Should Read This Guide

The audience is primarily Java application developers, Customer Information Control System (CICS) application developers, Information Management System (IMS) application developers, and system administrators who configure gateway software on BEA Web Logic Server (WLS) platforms.

System Administrators

As the application administrator, you install the WLS platform software and configure the JAM gateway. You must have sufficient System Network Architecture (SNA) knowledge to configure the underlying SNA stack (PU2.1 server) so it conforms with definitions created in Virtual Telecommunications Access Method (VTAM) and CICS for each remote domain. Refer to stack vendor documentation for details.

Successfully linking and establishing conversations between Java applications and mainframe-based programs requires special coordination. The names and characteristics of mainframe resources, configured in the SNA stack, must agree with resources and characteristics defined in VTAM and CICS.

Typically, remote VTAM and CICS resources are defined by system personnel in a data center where IBM mainframes are located. Therefore, you need to request the remote names of IMS and CICS resources from data center systems personnel and use those names to configure the SNA stack and the JAM gateway.

Java Application Developers

The JAM facility provides application developers bidirectional, transparent processing using platform native Application Programming Interfaces (API). Java application programmers can develop clients or servers using standard object-oriented programming techniques without regard for mainframe protocols.

CICS Application Developers

CICS programmers can develop servers or clients using the CICS Link command and the Distributed Program Link (DPL) subset API as a standard distributed CICS application. In addition, any existing CICS application designed to be invoked from a DPL can be used without modification.

IMS Application Developers

IMS programmers can use any IMS program (without modification) that sends and receives messages to and from the IMS message queue as either a client or a server.

How this Guide Is Organized

The *BEA eLink Java Adapter for Mainframe WLS Edition User Guide* is organized as follows:

- **Introducing eLink Java Adapter for Mainframe WLS Edition**

This section gives an overview of the JAM environment and describes two configurations for the product.

- **Configuring the Java Adapter for Mainframe Environment**

This section provides information and procedures for configuring the JAM development environment to create a basic servlet that can execute a CICS DPL.

- **Developing Java Applications**

This section explains how to build a base Java application from a COBOL copybook and generate Java application source code.

- **Deploying Applications**

This section gives procedures for installing developed servlets and/or EJB into an operational environment.

- **Security**

This section describes the basic level of security supported by the JAM product software.

- **Programming Reference**

This section provides data mapping rules for the `EgenCobol` tool used to generate Java code from COBOL copybooks.

- Programming Scenarios

This section provides three programming scenarios for developing and deploying JAM applications. You can follow the steps to learn practical uses for the tools included with the JAM product.

In addition, the appendixes contain the following information:

- Code Generator Reference Pages

This appendix contains reference pages for the `EgenCobol` tool.

- Configuration Checker Utility

This appendix describes the configuration checker utility that can be used to verify the contents of the `jcrmgw.cfg` file before starting the Java Communications Resource Manager Gateway (JCRMGW).

- Error and Informational Messages

This appendix gives descriptions of messages that may be encountered while using JAM software.

- Java Docs

This appendix tells how to use the HTML pages describing JAM Java classes.

- Glossary

The glossary contains definitions of some important terms used in this guide.

Product Documentation

The JAM documentation consists of the following:

- *BEA eLink Java Adapter for Mainframe WLS Edition Release Notes*
- *BEA eLink Java Adapter for Mainframe WLS Edition Installation Guide*
- *BEA eLink Java Adapter for Mainframe WLS Edition User Guide*
- *BEA eLink Adapter for Mainframe, SNACRM Administration Guide*

How to Use The Documentation

The Documentation CDROM included in the package with your product software CDROM contains an HTML Web User Interface (WUI). The WUI links to HTML versions and PDF versions of JAM documentation. The WUI should be viewed on an online browser. The PDF versions should be used for printing. (Information on how to view the online documentation is available in the *BEA eLink Java Adapter for Mainframe WLS Edition Release Notes*.)

Note: The WUI requires a Web browser that supports HTML 3.0. Netscape Navigator 2.02 or Microsoft Internet Explorer 3.0 or later.

You must have the Adobe Acrobat Reader to print the PDF file. If you do not have this reader, you can obtain it free of charge from the Adobe Systems Incorporated home site at www.adobe.com. (The BEA eLink Java Adapter for Mainframe WLS Edition WUI contains a hot link to this site.)

Document Conventions

The following documentation conventions are used throughout this manual:

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.

Convention	Item
monospace text	<p>Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.</p> <p><i>Examples:</i></p> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	<p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void commit ()</pre>
<i>monospace italic text</i>	<p>Identifies variables in code.</p> <p><i>Example:</i></p> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	<p>Indicates device names, environment variables, and logical operators.</p> <p><i>Examples:</i></p> <pre>LPT1 SIGNON OR</pre>
{ }	<p>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</p>
[]	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f <i>file-list</i>]... [-l <i>file-list</i>]...</pre>
	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>

Convention	Item
...	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none"> ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
. . .	<p>Indicates the omission of items from a code example or from a syntax line.</p> <p>The vertical ellipsis itself should never be typed.</p>

Contact Us

If you have any questions about this version of BEA eLink Java Adapter for Mainframe WLS Edition, or if you have problems installing and running the software, contact BEA Customer Support through BEA WebSupport at www.beasys.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the products you are using
- A description of the problem and the content of pertinent error messages



1 Introducing eLink Java Adapter for Mainframe WLS Edition

This section covers the following topics:

- JAM Environment
- Configuration Options

JAM Environment

The BEA eLink Java Adapter for Mainframe WLS Edition (JAM) environment is a set of software components that provides seamless bidirectional interactions between Java applications running on a WebLogic Server (WLS) platform and either Customer Information Control System (CICS) applications, or Information Management System (IMS) applications running on a mainframe. With CICS, the Java application request/response operations interact using Distributed Program Links (DPL). With IMS, the Java application request/response operations interact using IMS implicit Application Program-to-Program Communication (APPC) support.

This guide provides you with scenarios for developing and deploying applications by using the software components within the JAM environment. The scenarios depict specific tasks that work in conjunction with development and deployment procedures also provided in this guide. It is best to read and become familiar with the contents of “Developing Java Applications” and “Deploying Applications” before attempting the exercises in “Programming Scenarios.”

To learn more about how the System Network Architecture Communications Resource Manager (SNACRM) provides the emulation that enables CICS DPL protocols to flow into and out of the JAM environment, refer to the *BEA eLink Adapter for Mainframe, SNACRM Administration Guide*. This guide is accessible from the JAM documentation Web User Interface (WUI) in both HTML and PDF formats.

The JAM environment includes the following components:

- WebLogic Server (WLS)
- System Network Architecture Communications Resource Manager
- Java Communications Resource Manager Gateway (JCRMGW)
- EgenCobol Code Generator
- Supported Third-Party SNA Stack

Note: The WLS and third-party SNA stack are required as parts of the JAM environment, but are sold separately.

WebLogic Server

The WLS application server provides the environment for running Java Servlets and Enterprise Java Beans (EJB). The Java gateway is launched during the WLS startup using a startup class called `gwboot`. Configuration requirements for the server to start the gateway are limited to identifying the startup class along with any start-up arguments in the `weblogic.properties` file.

Note: Although JAM can be used with WebLogic Express (WLX), no EJB capability would be available because WLX does not support it.

System Network Architecture Communications Resource Manager

The SNACRM runs as a separate native process. It enables APPC conversations and DPL protocols to flow into and out of the Java environment and runs on the same platform as the SNA stack. The SNACRM obtains its configuration from the JCRMGW. If the Java gateway is running on a platform other than the one on which the SNACRM is running, the SNACRM should be started before the WLS is started so it is monitoring the address specified in the JCRMGW configuration.

The SNACRM supports non-transactional IMS programs using the implicit APPC support for IMS. Implicit APPC is similar to the CICS/ESA DPL. Any IMS program that sends and receives messages to and from the IMS message queue can be used without change as either a client or a server.

In order for the SNACRM to properly operate within the JAM environment, the CICS remote domain must be prepared to conduct operations with the local domain. This includes:

- Establishing the Virtual Telecommunications Access Method (VTAM) configuration
- Configuring the CICS Local Unit (LU)
- Completing cross-platform definitions
- Setting stack traces

Refer to the *BEA eLink Adapter for Mainframe, SNACRM Administration Guide* for more detailed information. That guide is accessible from the JAM documentation WUI in both HTML and PDF formats.

Java Communications Resource Manager Gateway

The JCRMGW is a Java application that manages sessions providing access into and out of the Java environment. It is configured using a text file named `jcrmgw.cfg` residing in the WLS directory. Requests coming from the mainframe are mapped to an

EJB which services the request while requests going to the mainframe are mapped to a mainframe program which can be executed using a CICS DPL or started from an IMS queue.

EgenCobol Code Generator

The EgenCobol code generator is a utility that generates Java source code from a COBOL copybook. A base Java application is provided on the JAM product CDROM so you can enhance it to create custom Java applications. The generated code is dependent on the generation model chosen:

- Servlet only
- Client EJB (with optional servlet)
- Server EJB
- Java Class

The generated code provides for data accessors and conversions as well as miscellaneous functions. The generated servlet provides a basic form that matches the copybook. You can use this servlet for testing or proof of concept. For examples of using the EgenCobol code generator, refer to “Developing Java Applications” and “Programming Scenarios.”

Supported Third-Party SNA Stack

A properly configured SNA protocol stack is required for the SNACRM to communicate with a mainframe. (Refer to the *BEA eLink Java Adapter for Mainframe WLS Edition Release Notes* for a complete list of supported stacks.) The JCRMGW requires several parameters from the SNA stack configuration:

- Local LU

The local name of the SNACRM that is used by the mainframe CICS region.

- Remote LU

An alias representing the mainframe partner application to be used by the SNACRM.

- Mode name

SNA mode definitions provide session characteristics for given local/remote LU pair.

- MAX Sessions (optional)

Sessions are required for connections to be made between two LU. MAX sessions should be high enough to handle the expected gateway traffic.

- Minimum Contention Winners (optional)

This parameter is optional and determines priority of session activation.

Configuration Options

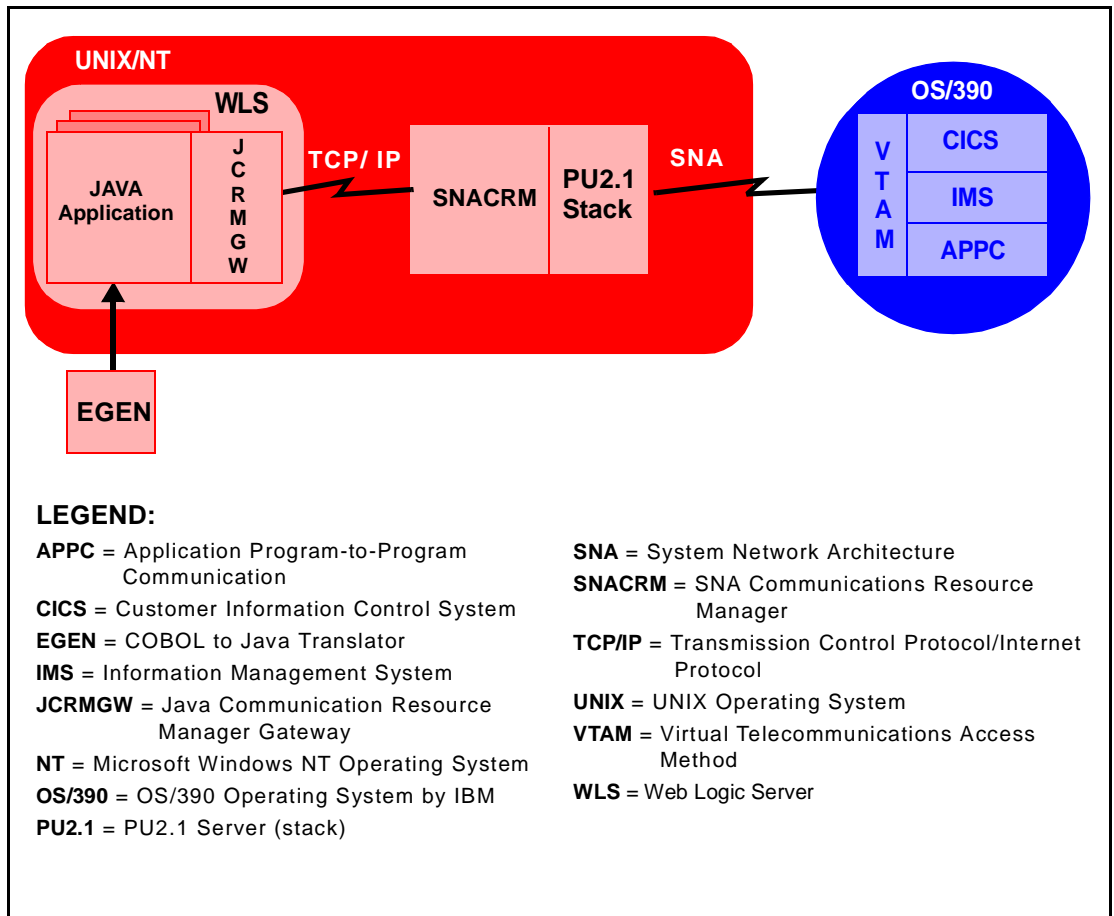
Figure 1-1 and Figure 1-2 show the component relationships of the JAM environment in two configurations, the *combined configuration* and the *distributed configuration*. This section contains the following topics:

- Combined Configuration
- Distributed Configuration
- Installation Considerations

Combined Configuration

This configuration combines the Java application, JCRMGW, WLS, and the SNACRM with the stack (PU2.1 server) on the same UNIX or Windows NT platform. It employs the IBM proprietary SNA protocol for transactions with the mainframe via the stack (see Figure 1-1).

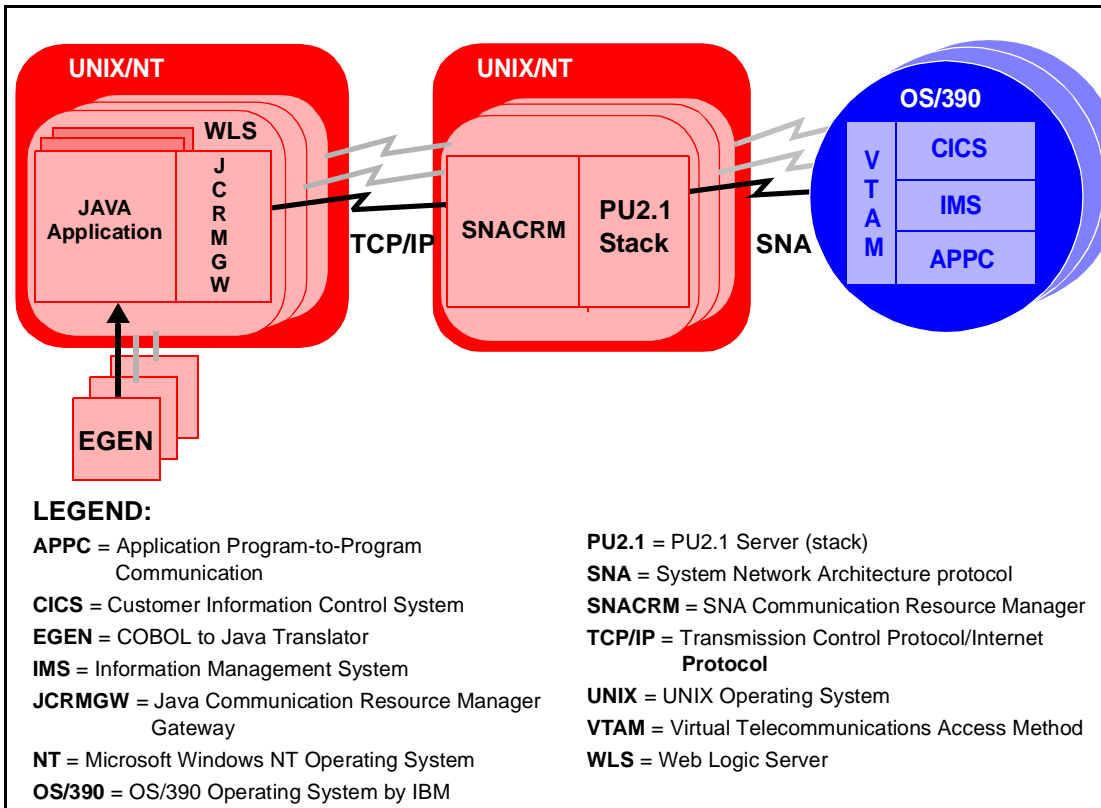
Figure 1-1 JAM Combined Configuration



Distributed Configuration

The second configuration separates the Java applications and JCRMGW from the SNACRM on different UNIX or Windows NT platforms. It employs the widely used Transmission Control Protocol/Internet Protocol (TCP/IP) connectivity between the Java applications platform and the SNACRM platforms, as well as the SNA connectivity to the mainframe environment(s). This configuration gives you the flexibility to deploy WLS separately from the SNACRM at installations that require WLS on a platform other than the one on which the SNA stack is running. Note that this configuration requires a one-to-one relationship between the local JCRMGW and the remote SNACRM (see Figure 1-2).

Figure 1-2 JAM Distributed Configuration



Installation Considerations

It is important to understand which of the configurations applies to your application development environment. During installation, you must enter the destinations for the two major JAM components, the SNACRM and the JCRMGW. Those destinations are determined by your configuration. If it is the combined configuration, both components are installed on the same platform. If it is the distributed configuration, the components are installed on different platforms; the JCRMGW must be on the platform with WLS and the SNACRM on the same platform with the SNA stack. Refer to “Pre-Installation Considerations” in the *BEA eLink Java Adapter for Mainframe WLS Edition Installation Guide* for additional information on this subject.

2 Configuring the Java Adapter for Mainframe Environment

The BEA eLink Java Adapter for Mainframe WLS Edition (JAM) software provides application developers bidirectional, transparent processing using native platform Application Programming Interfaces (APIs). Java application programmers can develop clients or servers using standard object oriented programming techniques without regard for mainframe protocols.

Customer Information Control System (CICS) programmers can develop servers or clients using the CICS Link command and the Distributed Program Link (DPL) subset API as a standard distributed CICS application. In addition, any existing CICS application designed to be invoked from a DPL can be used without modification. Implicit APPC support can be used to provide similar capabilities for IMS.

This section outlines the steps required to configure the JAM development environment to create a basic servlet that can execute a CICS DPL.

Configuring the JAM Environment

Prior to producing working Java code, it is recommended that you configure the JAM environment in this order:

1. Install and configure a supported SNA stack.
2. Install Java JDK 1.1.7B or 1.2 for the WebLogic Server platform.

Obtain the JDK from the OEM who provided the hardware on which WebLogic Server is to run. Follow the installation instructions provided with the JDK distribution.

3. Install WebLogic Server.
4. Install the Java Adapter for Mainframe.
5. Install the SNA Communications Resource Manager.
6. Establish Your Mainframe Environment.
7. Configure the Java Communication Resource Manager Gateway.
8. Start WebLogic Server.

Step 1 - Install a Supported SNA Stack

A properly configured SNA protocol stack is required for the SNACRM to communicate with a mainframe. The Java gateway requires the following parameters from the SNA stack configuration:

- Local LU - The local name of the SNACRM that will be used by the mainframe application.
- Remote LU - The name of the mainframe application that the SNACRM will use.
- Mode Name - The SNA mode definitions provide session characteristics for a given local/remote LU pair.

- **Max sessions** - Sessions are required for connections to be made between two LU. Max sessions should be high enough to handle the expected gateway traffic. This parameter is optional and defaults to 4.
- **Minimum Contention Winners** - Determines the priority of the session activation. This parameter is optional and defaults to one half the value specified for max sessions.

Refer to the *BEA eLink Java Adapter for Mainframe WLS Edition Release Notes* for a list of supported SNA stacks.

Note: Refer to the SNA stack vendors documentation on how to configure for your environment. Ensure the stack is properly installed and the configuration can be activated.

Step 2 - Install the Java JDK

Obtain the JDK from the OEM who provided the hardware intended to run the WebLogic server. Follow the installation instructions provided with the JDK distribution.

Step 3 - Install WebLogic Server

The WebLogic application server provides the environment for running Java servlets and Enterprise Java Beans. The JCRMGW gateway is launched during the WLS startup using a startup class called `gwboot`. Configuration requirements for the server to start the gateway are limited to identifying the startup class along with any startup arguments in the `weblogic.properties` file. This file is located in the WebLogic installation directory generated for it.

The only WLS configuration requirement for the JCRMGW is in the `weblogic.properties` file and the `WEBLOGICCLASSPATH` environment variable, which is set in the `start.weblogic` startup script found in the WLS installation directory. The `WEBLOGICCLASSPATH` also requires the addition of the `jam.jar` file. In addition, the `weblogic.properties` file requires an entry for the `gwboot` startup class that launches the gateway when the server is started.

2 *Configuring the Java Adapter for Mainframe Environment*

When the SNACRM is installed on a different machine from the JAM, you must specify the `-r` parameter on the `weblogic.system.startupArgs.jcrmgw` line of the `weblogic.properties` file. If the two JAM components are installed on the same machine, the `-r` parameter is optional since the gateway can spawn a SNACRM from the startup process.

The following listing contains an example of the entries to be added in the `weblogic.properties` file.

Listing 2-1 Sample `weblogic.properties` File

```
weblogic.system.startupClass.jcrmgw=bea.sna.jcrmgw.gwboot
weblogic.system.startupArgs.jcrmgw=[-t] [-r] [-u <userid>]
```

Where:

`-t`

will turn on tracing in the SNACRM.

`-r`

indicates that the SNACRM is remote. The SNACRM should be run on the same platform as the SNA stack. This can be the same machine as the WebLogic server, in which case `-r` can be omitted and the JCRMGW will spawn a new SNACRM using the address specified in the `jcrmgw.cfg` file. If `-r` is used, the SNACRM will not spawn, but is assumed to already be running at the address specified in the `jcrmgw.cfg`, even if this is the same machine running the gateway. In addition, the local SNACRM group identifier must match the group identifier specified on the command line used to start the remote SNACRM.

`-u<userid>`

`<userid>` is the mainframe userid that should be associated with all requests originating from this gateway. This is useful for IDENTIFY type security where the client cannot provide a userid.

Note: The `WEBLOGICCLASSPATH` must contain the fully-qualified name of the `jam.jar` file.

Step 4 - Install the Java Access for Mainframe

Refer to the *BEA eLink Java Adapter for Mainframe WLS Edition Installation Guide* and *Release Notes* for information on hardware and software requirements and instructions on installing the software.

Step 5 - Install the SNA Communications Resource Manager

The following environment variables must be set in the environment where the SNACRM is started.

```
FLDTBLDIR32=JAM Installation Directory/lib  
FIELDTBLS32=JAM Installation Directory/fmb.def  
APPDIR=<wherever>
```

These environment variables can be added as 'set' or 'export' commands in a script file used to start a SNACRM. Or they can be added to the `startweblogic` script for use when a SNACRM is spawned on startup.

Refer to the *BEA eLink Java Adapter for Mainframe WLS Edition Installation Guide* for information on hardware and software requirements and instructions on installing the software. For additional operational and administrative information, refer to the *BEA eLink Adapter for Mainframe SNACRM Administration Guide*.

Step 6 - Establish Your Mainframe Environment

The following configurations are required on the mainframe in order to conduct operations with the BEA TUXEDO/WLS environment:

- Established VTAM configuration
- CICS/ESA Logical Unit (LU) configured
 - Creating connections
 - Defining sessions
- If using IMS, APPC and transaction definitions must exist for that environment

Step 7 - Start WebLogic Server

Start WebLogic Server and verify that the *SnacTask Startup Complete* message indicates the gateway is ready. Also you can verify that the CICS connection is acquired using the CICS CEMTI CON (*) Command to verify your connection is in the ACQUIRED state, indicating the link is ready for use.

Step 8 - Configure the Java Communication Resource Manager Gateway (JCRMGW)

The JCRMGW uses a `jcrmgw.cfg` configuration file that you create to control much of the operation of the JCRMGW. The JCRMGW configuration file defines the SNACRM, stack, links, and local and remote services that comprise the gateway environment. This file must be created and located in the WebLogic installation directory.

The general format of the `jcrmgw.cfg` configuration file is as follows:

- The file is made up of six sections. Lines beginning with an asterisk (*) indicate the beginning of a specific section. Each such line contains the name of the section immediately following the *.
- Parameters are generally specified by: *KEYWORD* = *value*. This sets *KEYWORD* to *value*. Valid keywords are described in the following sections. *KEYWORDS* are reserved. They cannot be used as *values* unless they are quoted.
- Input fields are separated by at least one space (or tab) character.
- “#” introduces a comment. A new line ends a comment.
- Blank lines and comments are ignored.
- Comments can be freely attached to the end of any line.

The following paragraphs describe the significant parameters within specific sections of the `jcrmgw.cfg` file that define new gateway configurations.

JC_REMOTE_DOMAINS Section

This section provides an alias for associating mainframe applications with services and links. Any given entry in this section must be named. A name is provided by using a label on any one of the keywords that comprise the entry. Code examples will use a label on the first keyword of a given entry.

Required Keywords

The following are required keywords.

DOMAINID=<string>

<string> is any name to be used for identifying a partner system. This keyword/value pair is required and has no default value.

The following listing contains an example.

```
CRCICS1 DOMAINID="TestCICS"
```

JC_SNACRM Section

This section identifies the SNACRM that this gateway talks to. There is one SNACRM for each JCRMGW. Any given entry in this section must be named. A name is provided by using a label on any one of the keywords that comprise the entry. Code examples will use a label on the first keyword of a given entry.

Required Keywords

The following are required keywords.

SNACRMADDR=<string>

<string> is a symbolic tcp address in the form of: “//hostname:port”

Host name is the name of the machine that runs the SNACRM. Port is an unused decimal port number that the SNACRM uses to talk to the Java gateway. In the case of a SNACRM that is started independently of the gateway, this address must match the address used on the SNACRM command line. When the gateway is started, it tries to contact the SNACRM at the address.

The following listing contains an example.

```
MYSNACRM SNACRMADDR="//dalhp55:6942"
```

The gateway will look for a SNACRM on a machine named dalhp55 at port 6942.

GROUP=<string>

<string> is any name to be used to correlate a gateway with a SNACRM. The name is used as part of the file name for the SNACRM logs. In the case of a SNACRM that is started independently of the gateway, this name must match the group name used on the SNACRM command line, even if the default name is used. The keyword/value pair is optional and has a default value of “SNAGROUP”.

The following listing contains an example.

```
GROUP="CRAUTH"
```

The SNACRM expects a gateway signon for group CRAUTH.

JC_SNASTACKS Section

This section identifies the Local LU used for the SNACRM along with the stack identifier for the SNA stack library to be used. Only one local LU and one stack can be specified for a SNACRM. Any given entry in this section must be named. A name is provided by using a label on any one of the keywords that comprise the entry. Code examples will use a label on the first keyword of a given entry.

Required Keywords

The following are required keywords.

STACKTYPE=[hp60 | ibm50 | ibm60 | ms40 | spx60 | sun91]

One of the specified tokens must be used. These names determine which SNA stack support library will be loaded.

The following stacks can be identified:

- hp60 = Hewlett Packard SNA 2 Plus 6.0 on HP-UX
- ibm50 = IBM Communications server 5.0 on WINNT or AIX
- ibm60 = IBM Communications server 6.0 on WINNT
- ms40 = Microsoft SNA Server 4.0 SP3 on WINNT
- spx60 = Data Connection Snapix 6.0 on Solaris
- sun91 = Sunlink 9.1 on Solaris

The keyword/value pair is required and has no default value.

The following listing contains an example.

```
ibmcsaix STACKTYPE="ibm50"
```

The SNACRM tries to load the library for IBM Communication Server 5.0.

LOCALLU=<string>

<string> is an alias for the local LU to be used by the SNACRM. This must match a corresponding LU alias defined in the SNA Stack configuration. This alias may or may not match the actual LU name. This keyword/value pair is required and has no default value.

The following listing contains an example.

```
LOCALLU="AUTHAPP"
```

The SNACRM tries to use the local LU which has been defined in the SNA stack configuration with an alias of AUTHAPP.

JC_SNALINKS Section

This section identifies partner mainframe application regions. Multiple links for a single SNACRM are supported. Any given entry in this section must be named. A name is provided by using a label on any one of the keywords that comprise the entry. Code examples will use a label on the first keyword of a given entry.

Required Keywords

The following are required keywords.

RDOM=<string>

<string> is a name previously defined as a remote domain. The remote domain naming is a mechanism for grouping links. This keyword/value pair is required and has no default value.

The following listing contains an example.

```
RDOM="CRCICSL1"
```

This link is associated with the remote domain named CRCICSL1.

RLUNAME=<string>

<string> is an alias for the remote LU representing the partner application to be used by the SNACRM. This must match a corresponding partner/remote LU alias defined in the SNA Stack configuration. This alias may or may not match the actual remote LU name. This keyword/value pair is required and has no default value.

The following listing contains an example.

```
RLUNAME="AUTHAPPL"
```

The SNACRM routes all traffic for this link to a remote application defined with an alias of AUTHAPPL. This alias must be defined in the SNA stack configuration for a valid mainframe application name.

MODENAME=<string>

<string> is the name of a mode definition to be used for applications on this link. This must match a corresponding mode name defined in the SNA Stack configuration and the VTAM mode table. A valid mode name is provided by mainframe support personnel. This keyword/value pair is required and has no default value.

The following listing contains an example

```
MODENAME="SMSNA100"
```

The SNACRM will use the SMSNA100 mode for applications on this link.

Optional Keywords

The following are optional keywords.

MAXSESS=*nn* (4)

nn is the maximum number of SNA sessions that can be started for this link. The actual value used is negotiated with the partner and can be lower than this value if the partner is configured with a lower value. This value includes two sessions for the SNA Service manager mode. The lowest usable value is 4; it provides two sessions for the application and two sessions for the service manager. This keyword/value pair is optional and has a default value of 4.

The following listing contains an example.

```
MAXSESS=16
```

The maximum number of sessions on this link is set to 16 for all modes combined.

MINWIN=*nn* (MAXSESS/2)

nn is the minimum number of SNA sessions that will be contention winners for the SNACRM. This keyword is not required and defaults to one half the number of MAXSESS. In most cases this is suitable unless asymmetric winners are desired due to application requirements. Also, the default value may not be appropriate if the maximum is negotiated down to a value lower than half of the specified maximum. This value includes one session for the SNA Service Manager mode. In general, the lowest practical value is 2, which provides one session for the application and one session for the service manager. See IBM's SNA documentation for a complete discussion of session limits and contention winners. The keyword/value pair is optional and has a default value of half the value of MAXSESS.

The following listing contains an example.

```
MINWIN=8
```

The number of contention winner sessions on this link is set to 8 for all modes combined.

`SECURITY=[LOCAL | IDENTITY | VERIFY] (LOCAL)`

This is an optional keyword. If entered, one of the specified tokens must be used, and the mainframe connection be configured to match.

The meaning of the values are:

LOCAL = All security is handled by the local system and the link itself has no security requirement.

IDENTIFY = A userid will be passed on to the mainframe. This userid can originate with the client application or it can be a default userid supplied with the `-u` option on the gateway startup.

VERIFY = A userid and password will be passed onto the mainframe. The userid can originate with the client application or it can be a default userid supplied with the `-u` option on the gateway startup. The password must be supplied by the client application.

This keyword/value pair is optional and has a default value of **LOCAL**.

The following listing contains an example.

```
SECURITY=IDENTIFY
```

A userid is required for all requests made on this link. Note that the mainframe configuration for the remote LU (*i.e.*, a CICS connection definition) must have a security level that matches.

JC_LOCAL_SERVICES Section

This section maps incoming mainframe program names to a Home interface for an EJB that will service the request. Any given entry in this section must be named. A name is provided by using a label on any one of the keywords that comprise the entry. Code examples will use a label on the first keyword of a given entry.

Required Keywords

The following are required keywords.

```
RNAME=<RRRRRRR>
```

Rname is the remote resource name associated with this service. For a CICS Distributed Program Link, this is the actual program name that was invoked from the mainframe. For a DTP style request, the resource name must conform to the TP ID requirements of the originating system. A CICS DPL Program and an IMS Transaction ID are limited to eight characters.

The label on this entry must be the name of a valid home interface for an Enterprise Java Bean registered with JNDI. This keyword/value pair is required and has no default value.

The following listing contains an example.

```
StatelessSession.TraderHome RNAME="DPL1SVR"
```

DPL1SVR is the name of the program that was invoked from the mainframe and `StatelessSessions.TraderHome` is the name of the home interface which will be used to invoke the EJB that will service this request.

JC_REMOTE_SERVICES Section

This section maps remote mainframe program names to method names that can be used by a local application to invoke the remote request. Any given entry in this section must be named. A name is provided by using a label on any one of the keywords that comprise the entry. Code examples will use a label on the first keyword of a given entry.

Required Keywords

The following are required keywords.

RDOM=<string>

<string> is a name previously defined as a remote domain. This keyword/value pair is required and has no default value.

The following listing contains an example

```
RDOM="CRCICS1"
```

This service will be associated with the remote domain named CRCICS1.

RNAME=<[MMMM:]RRRRRRRR>

Rname is the remote resource name associated with this service. For a CICS Distributed Program Link, this is the actual program name to be invoked. The first portion of this value (MMMM:) is optional and can be an alternate mirror transaction identifier. An alternate mirror transaction is useful for some mainframe database deployments as well as security or charge back systems. The alternate mirror transaction cannot exceed 4 characters in length. The second portion (RRRRRRRR) is the program name to invoke for a CICS DPL. For a DTP style request, the resource name must conform to the Tran ID requirements of IMS.

The following listing contains an example.

```
RNAME= "AUTH:DPLQRY"
```

This Rname is a program named DPLQRY and will use an alternate mirror transaction name of AUTH.

Optional Keywords

The following are optional keywords.

FUNCTION=[DTP | DPL] (DPL)

This is an optional keyword used to indicate the method of request invocation accepted by the mainframe. DPL is the default and is only valid when used with a CICS partner and an application that can be invoked using a Distributed Program Link. DTP should be used when invoking services from an IMS server.

This keyword/value pair is optional and has a default value of DPL.

The following listing contains an example.

```
FUNCTION=DTP
```

This remote service will be invoked as an IMS application rather than a program link.

TRANTIME=nnn (30000)

nnn is the maximum number of milliseconds the client application will block before a host request is timed out. This keyword/value pair is optional and has a default value of 30000.

The following listing contains an example.

```
TRANTIME=120000.
```

This remote service will time out if the mainframe does not respond within 2 minutes (120 seconds).

Sample Configuration File

The following example illustrates a basic `jcrmgw.cfg` file.

Listing 2-2 Sample `jcrmgw.cfg` Configuration File

```
*JC_REMOTE_DOMAINS
#
CICS410          DOMAINID="410"
#
*JC_SNACRM
#
SNACRMAN         SNACRMADDR="//daInt66:8650"
                  GROUP="SNAG1"

*JC_SNASTACKS
#
IBMCSAIX         STACKTYPE="IBM50"
                  LOCALLU="LUNT66B"
#
*JC_SNALINKS
#
C41XB01          RLUNAME="C410XB01"
                  RDOM="CICS410"
                  MODENAME="SMSNA100"
                  MAXSESS=8
                  MINWIN=6
#
*JC_LOCAL_SERVICES
#
TraderHome       RNAME="DPL1SVR"
#
JC_REMOTE_SERVICES
DPLINIT          RDOM="CICS410"
                  RNAME="PRIM:DPLINIT"
                  TRANTIME=10000
TOUPPER          RDOM="CICS410"
                  RNAME="TOUPDPLS"
                  TRANTIME=10000
```

2 *Configuring the Java Adapter for Mainframe Environment*

demoRead	RDOM="CICS410 " RNAME="DPLDEMOR " TRANTIME=10000
demoUpdate	RDOM="CICS410 " RNAME="DPLDEMOU " TRANTIME=10000
demoCreate	RDOM="CICS410 " RNAME="DPLDEMOC " TRANTIME=10000
demoDelete	RDOM="CICS410 " RNAME="DPLEMOD " TRANTIME=10000
imsInsert	RDOM=" IMS51 " FUNCTION=DTP RNAME=" IMSSVR12 "

3 Developing Java Applications

The BEA eLink Java Adapter for Mainframe WLS Edition (JAM) code generation tool produces Java code from COBOL copybook source files. This generated Java code consists of both data mapping code and framework code for any application that will invoke the mainframe application through the gateway.

The following sections describe how the base Java application is generated, what role the COBOL copybook plays in the generation, and how to determine the type of application you want to generate.

This topic consists of the following sub-topics.

- Building the Base Java Application
- Obtaining the COBOL Copybook
- Generating the Java Application Source

Note: Some of the code sample listings in this topic have field names in bold for easier reading. Also, Comment-numbered items have corresponding comments at the bottom of each script example.

Building the Base Java Application

The JAM code generation tool allows you to produce working Java code that functions within a Java Server execution model. The Java code produced is centered around typed data buffers.

In COBOL/CICS terms, a typed data buffer represents a group data item, or record, defined by a COBOL copybook that specifies the COMMAREA data area used by a CICS DPL program.

In Java terms, a typed data buffer is an object (instantiation) of a class type. The fields of the class correspond to the data fields within a record. Field values are retrieved and modified through the use of accessor functions (i.e., get and set functions).

Data View Concepts

Typed data buffers are handled by the Java `DataView` class, which performs all of the necessary conversions and translations of the data fields within each record. The result is the ability to send and receive data buffers between Java clients and COBOL/CICS DPL programs. All of the underlying data conversions are performed transparently to each side of the application; the Java side of the application manipulates the data buffers as Java objects, and the COBOL/CICS side manipulates them as COMMAREA linkage section data items.

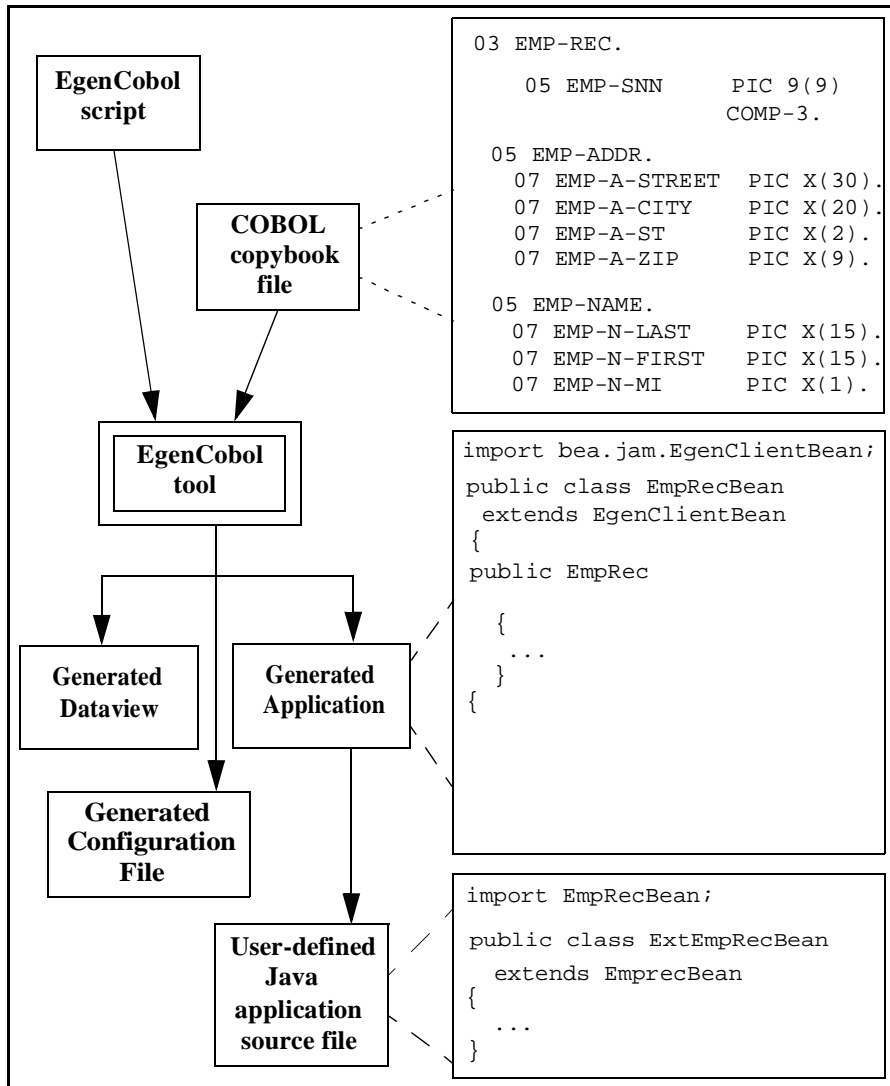
These Java files are then used by application programmers to write other Java source files that extend the classes defined in the generated source files. The user-defined classes typically add methods and variables that embody the business logic of the application.

The generated source files provide a simple framework centered around the typed data buffers (i.e., the COMMAREAs) of the CICS application. It is up to the Java application programmer to extend this framework of classes.

The heart of a JAM application is the sending and receiving of data records between the Java application system and the remote (mainframe) system. This is accomplished by employing a `DataView` that corresponds to a mainframe data record. On the mainframe side, the `DataView` represents the CICS COMMAREA data that is sent between DPL programs. On the Java side, the `DataView` represents a class object containing the COMMAREA data fields. The JAM application framework provides all of the data conversions necessary to send and receive data records between Java systems and the mainframe.

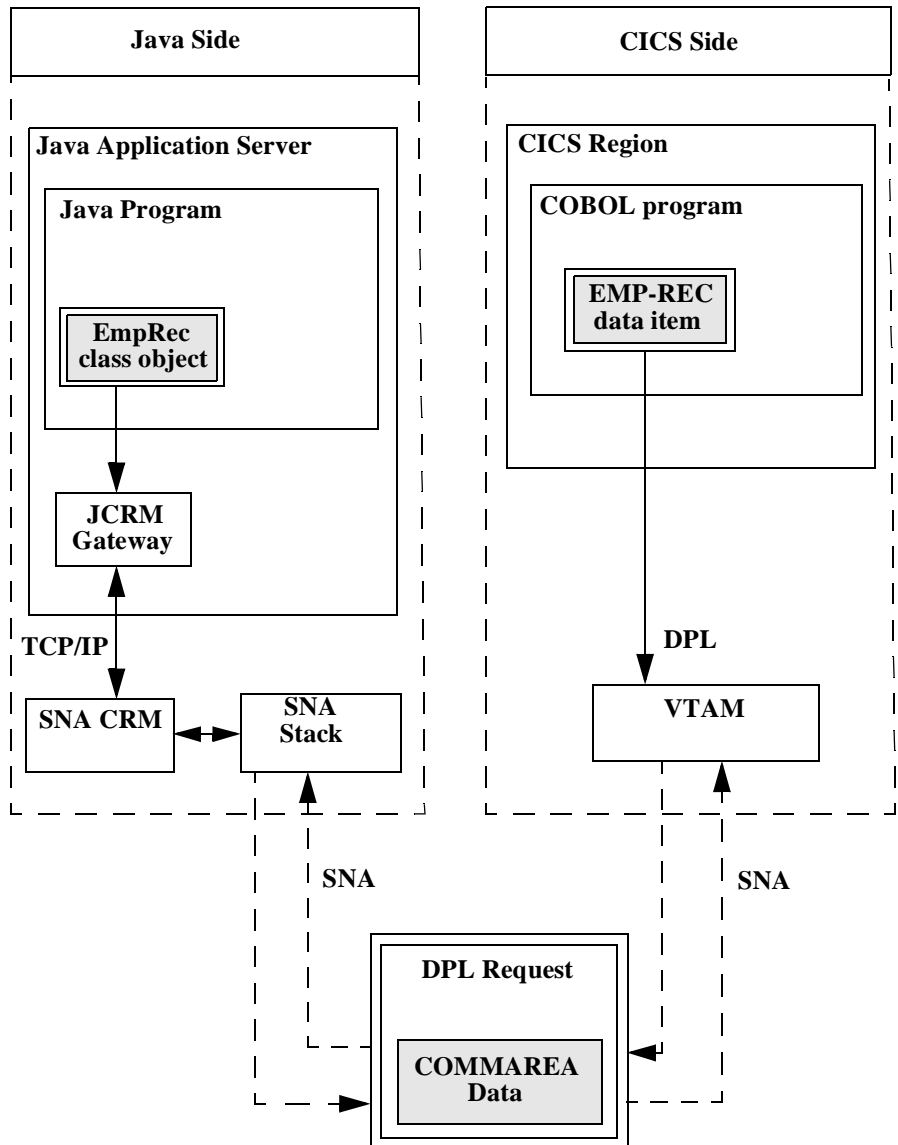
The following figure provides an illustration of how the underlying JAM generation tools work together.

Figure 3-1 Understanding the Underlying JAM Tools



The following figure provides an illustration of the JAM general application transport method.

Figure 3-2 JAM General Application Transport Method



Obtaining the COBOL Copybook

A mainframe COBOL/CICS, IMS, or mainframe application typically uses a copybook source file to define its COMMAREA. This file is specified in a COPY directive within the LINKAGE SECTION of the source program.

If the CICS application does not use a copybook file (but simply defines the COMMAREA directly in the program source), you will have to create one from the definition contained in the program source.

The JAM code generation tools support most of the COBOL data types and data clauses, however, some obsolete constructs along with floating point data types are not supported. Also, a few COBOL features are not supported. For unsupported constructs, you will have to determine if the copybook can be used without modification. In most cases, unsupported items are treated as alphanumeric data types or are simply ignored.

Each copybook's contents (which define a COMMAREA record) are parsed by the EgenCobol application generator tool, producing DataView sub-classes that provide facilities to:

- Convert COBOL data types to and from Java data types. This includes conversions for mainframe data formats and code pages.
- Convert COBOL data structures to and from Java data structures.
- Provide tools that may be used to convert the provided data structures into other arbitrary formats.

Creating a New Copybook

If you are producing a new application on the mainframe or modifying one that did not previously support DPL, then one or more new copybooks are required. You should keep in mind the COBOL features and data types that are supported by JAM as you create these copybooks.

Using an Existing Copybook

When a mainframe application has an existing DPL interface, it most likely has the data for that interface described in a COBOL copybook. The first thing that must be done is to verify that no COBOL features or data types that JAM does not support are used in the interface. The easiest way to check a copybook is to attempt to process it.

An example COBOL copybook source file is shown in the following listing. It is assumed that the copybook is identical to the copybook used by the mainframe COBOL application programs.

Listing 3-1 Sample `emprec.cpy` COBOL Copybook

```
1      *-----
2      * emprec.cpy
3      *      An employee record.
4      *-----
5
6      02      emp-record. (Comment 1)
7
8              04      emp-ssn                      pic 9(9)  comp-3.
9
10             04      emp-name.
11                 06      emp-name-last  pic x(15). (Comment 2)
12                 06      emp-name-first pic x(15).
13                 06      emp-name-mi   pic x.
14
15             04      emp-addr. (Comment 3)
16                 06      emp-addr-street pic x(30).
17                 06      emp-addr-st    pic x(2).
18                 06      emp-addr-zip   pic x(9).
19
20      * End
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-1 Script Comments

Comment 1	Declaration of a record (group) data item.
Comment 2	An elementary item.
Comment 3	An aggregate item.

The `emprec.cpy` copybook shown in the example declares a group item named `emp-record`. (This copybook is available in the `examples/samples.jar` file in JAM installation directory.

The JAM EgenCobol utility parses a COBOL copybook and generates a Java class that encapsulates the data record declared in the copybook. It does this by parsing an Egen script file containing a `DataView` definition similar to the following example.

Listing 3-2 Sample `emprec.egen` Script

```
view Sample.EmployeeRecord from emprec.cpy
```

The prior example specifies that the COBOL copybook file named `emprec.cpy` is to be parsed, and that a Java source file named `EmployeeRecord.java`, containing the definition of class `EmployeeRecord` in package `sample`, is to be generated from it.

Assuming this script was saved into a file named `emprec.egen`, the following shell command parses the copybook file named `emprec.cpy` and attempts to generate a Java source file named `EmployeeRecord.java` in the current directory:

Listing 3-3 Sample Copybook Parse Command

```
java bea.jam.egen.EgenCobol emprec.egen
```

If no error or warning messages are issued, the copybook is compatible with JAM and the source is created.

Note: You must establish a Java class path that refers to the JAM distribution jar file for this command to work properly. Please refer to the “Error Messages” section of this document for suggestions on resolving other problems.

The following example illustrates the resulting generated Java source file, `EmployeeRecord.java` with unimportant comments and implementation details removed for clarity.

Listing 3-4 `Generated EmployeeRecord.java Source File`

```
//EmployeeRecord.java
//Data view class generated by egencobol emprec.cpy

package Sample;(Comment 1)

//Imports

import bea.dmd.DataView.DataView;
...

/**DataView class for EmployeeRecord buffers*/

public final class EmployeeRecord (Comment 2)
    extends DataView
{
    ...

    // Code for field "emp-ssn"
    private BigDecimal    m_empSsn;(Comment 3)

    public BigDecimal    getEmpSsn() {...}(Comment 4)

    /** DataView subclass for emp-name Group */
    public final class EmpNameV (Comment 5)
        extends DataView
    {
        ...

        // Code for field "emp-name-last"
        private String    m_empNameLast;

        public void setEmpNameLast(String value) {...}
        public String getEmpNameLast() {...}(Comment 6)
```

```
// Code for field "emp-name-first"
private String m_empNameFirsrt;

public void setEmpnameFisrt (String value) {...}
public String getEmpNameFirst() {...}

// Code for field "emp-name-mi"
private String m_empNameMi;

public void setEmpNameMi (String value) {...}
public String getEmpnameMi() {...}
}

// Code for field "emp-name"
private EmpNameV m_empname" (Comment 7)

public EmpnameV getEmpname() {...}

/**DataView subclass for emp-addr Group */
public final class EmpAddrV
    extends DataView
{
    ...

    // Code for field "emp-addr-street"
    private String m_empAddrStreet;

    public void setEmpAddrStreet(Street value) {...}
    public String getEmpAddrStreet() {...}

    // Code for field "emp-addr-st"
    private String m_empAddrSt;

    public void setEmpAddrSt(String value) {...}
    public String getEmpAddrSt() {...}

    // Code for field "emp-addr-zip"
    private String m_empAddrZip;

    public void setEmpAddrZip(String value) {...}
    public String getEmpAddrZip() {...}
}

// Code for field "emp-addr"
private EmpAddrV m_empAddr;

public EmpAddrV getEmpAddr() {...}
}

//End EmployeeRecord.java
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-2 Script Comments

Comment 1	The package name is defined in the Egen script.
Comment 2	The data record is encapsulated in a class that extends the DataView class.
Comment 3	Each class member variable corresponds to a field in the data record.
Comment 4	Each data field has accessor functions.
Comment 5	Each aggregate data field has a corresponding nested inner class that extends the DataView class.
Comment 6	Each data field within an aggregate data field has accessor functions.
Comment 7	Each COBOL data field name is converted into a Java identifier.

Generating the Java Application Source

In addition to the COBOL copybook requirement, you must also determine which of the following application models you want to generate.

- Generating a Servlet-Only JAM Application
- Generating a Client Enterprise Java Bean-Based Application
- Generating a Server Enterprise Java Bean-Based Application
- Generating a Stand-Alone Client Application

The one you choose depends on the type of application you are building.

For all of the following applications you can generate, you must provide a script file containing definitions for the application, including the COBOL copybook file name, the DataView class names, and so forth.

Generating a Servlet-Only JAM Application

This type of application produces a Java servlet application that executes within a Java server environment (such as WebLogic Server). It is started from a web browser when the user enters a URL that is configured to invoke the servlet. The servlet presents an HTML form containing data fields and buttons. The buttons can be configured to invoke:

- EJB methods
- Remote gateway services (via the JCRMGW)

In general, servlets generated by EgenCobol are intended for testing purposes and are not easily customized to provide a more aesthetically pleasing interface.

Generating a Servlet-only JAM application consists of the following steps;

1. Creating a script.
2. Processing a script.
3. Working with the generated files.
4. Customizing the application.

The following topics discuss these steps and offer examples of each.

Creating a Script

In order to produce a servlet-only application, the script file that describes your DataViews must be enhanced to describe the mainframe services accessed, the browser pages produced, and the servlets that produce them. Service definitions look like the following listing.

Listing 3-5 Sample Service Definition

```
service getSalary
    accepts EmployeeRecord [cp037]
    returns EmployeeRecord [cp037]
```

This listing defines a service named `getSalary` that accepts an input buffer of type `EmployeeRecord` and returns an output buffer of type `EmployeeRecord`. Additional, the `EmployeeRecord` is transferred to and from the mainframe using character translation code page `cp037` (which is a U.S. EBCDIC character set). It is this service name which also requires an entry in the `jcrmgw.cfg` file.

A browser page that uses this service might be defined as the following.

Listing 3-6 Sample Page Definition

```
page EmployeeQuery "Employee Salary Query"
{
    view EmployeeRecord [cp037]

    buttons
    {
        Query service (getSalary)
            shows EmployeeQuery
    }
}
```

This listing defines an HTML page named `EmployeeQuery`, with a text title of “Employee Salary Query”, that displays an `EmployeeRecord` record object as an HTML form. It also specifies that the form has a button labeled “Query.” When the button is pressed, the service `getSalary` is invoked and is passed the contents of the browser page as an `EmployeeRecord` object (the fields of which may have been modified by the user). Afterwards, the `EmployeeQuery` page is used to display the results.

The servlet that serves this page might be defined like the following listing.

Listing 3-7 Sample servlet Definition

```
servlet EmployeeQueryServlet shows EmployeeQuery
```

This listing defines an application servlet class named `EmployeeQueryServlet`, and specifies that it displays the HTML page named `EmployeeQuery` as its initial display page.

The following listing shows a complete script for defining a servlet application.

Listing 3-8 Sample Servlet-Only Script

```
1  #-----
2  # empServlet.egen
3  #   JAM script for a servlet-only application.
4  #
5  #   $Id: empServlet.egen,v 1.2 2000/01/25 18:34:14 david Exp$
6  #-----
7
8  # DataViews (typed data records)
9
10 view sample.EmployeeRecord (Comment 1)
11     from emprec.cpy
12
13 # Services
14
15 service sampleCreate (Comment 2)
16     accepts EmployeeRecord
17     returns EmployeeRecord
18
19 service sampleRead (Comment 2)
20     accepts EmployeeRecord
21     returns EmployeeRecord
22
23 service sampleUpdate (Comment 2)
24     accepts EmployeeRecord
25     returns EmployeeRecords
26
27 service sampleDelete (Comment 2)
28     accepts EmployeeRecord
29     returns EmployeeRecord
30
31 # Servlet HTML pages
32
33 page initial "Initial page" (Comment 3)
34 {
35     view EmployeeRecord [CP037] (Comment 4)
36
37     buttons
38     {
```

```
39         "Create" (Comment 5)
40             service ("sampleCreate")
41             shows fullPage
42
43         "Read" (Comment 5)
44             service ("sampleRead")
45             shows fullPage
46     }
47 }
48
49 page fullPage "Complete page"
50 {
51     view EmployeeRecord
52
53     buttons
54     {
55         "Create"
56             service ("sampleCreate")
57             shows fullPage
58
59         "Read"
60             service ("sampleread")
61             shows fullpage
62
63         "Update"
64             service ("sampleDelete")
65             shows fullpage
66
67         "Delete"
68             service ("sampleDelete")
69             shows fullpage
70     }
71 }
72
73 # Servlets
74
75 servlet sample.SampleServlet (Comment 6)
76     shows initial
77
78 # End
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-3 Script Comments

Comment 1	Defines a DataView class, specifying its corresponding copybook source file and its package file.
Comment 2	Defines a service function and its input and output parameter types.
Comment 3	Defines an HTML page to be displayed by the servlet.
Comment 4	Specifies the DataView class to display on the page and the character code page.
Comment 5	Defines a button and its associated class method.
Comment 6	Defines a servlet class and its initial HTML display page.

Processing a Script

Issuing the following command will process the script:

Listing 3-9 Sample Script Process Command

```
%java bea.jam.egen.EgenCobol empservlet.egen
emprec.cpy, Lines: 21 Errors: 0, Warnings:0
Generating sample.EmployeeRecord...
Generating group emp-name
Generating group emp-addr
Generating sample.SampleServlet...
```

Generated Files

This script command generates the following files.

Table 3-4 Sample Script Generated Files

Files	Content
SampleServlet.java	Servlet source code.
EmployeeRecord.java	Source for the DataView object.

SampleServlet.java Source File

The following listing illustrates the contents of the generated SampleServlet.java source file (with some parts omitted).

Listing 3-10 Sample SampleServlet.java Contents

```
// SampleServlet.java
//
// Servlet class generated by egencobol on 25-Jan-2000.

package sample;

// Imports

import javax.servlet.http.HttpServlet;
import weblogic.html.ServletPage;
import bea.dmd.DataView.DataView;
import bea.jam.egen.EgenServlet;
...

/** servlet class for EmployeeRecord buffers. */

public class SampleServlet
    extends EgenServlet
{
    /** Create a new servlet. */
    public SampleServlet() (Comment 1)
    {
        ...
    }
}
```

```
        /** Get an instance of the initial DataView for this
        Servlet.*/
        protected DataView initialDataView()
        {
            ...
        }

        ...
    }

//End SampleServlet.java
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-5 Script Comments

Comment 1	Defines a servlet class.
-----------	--------------------------

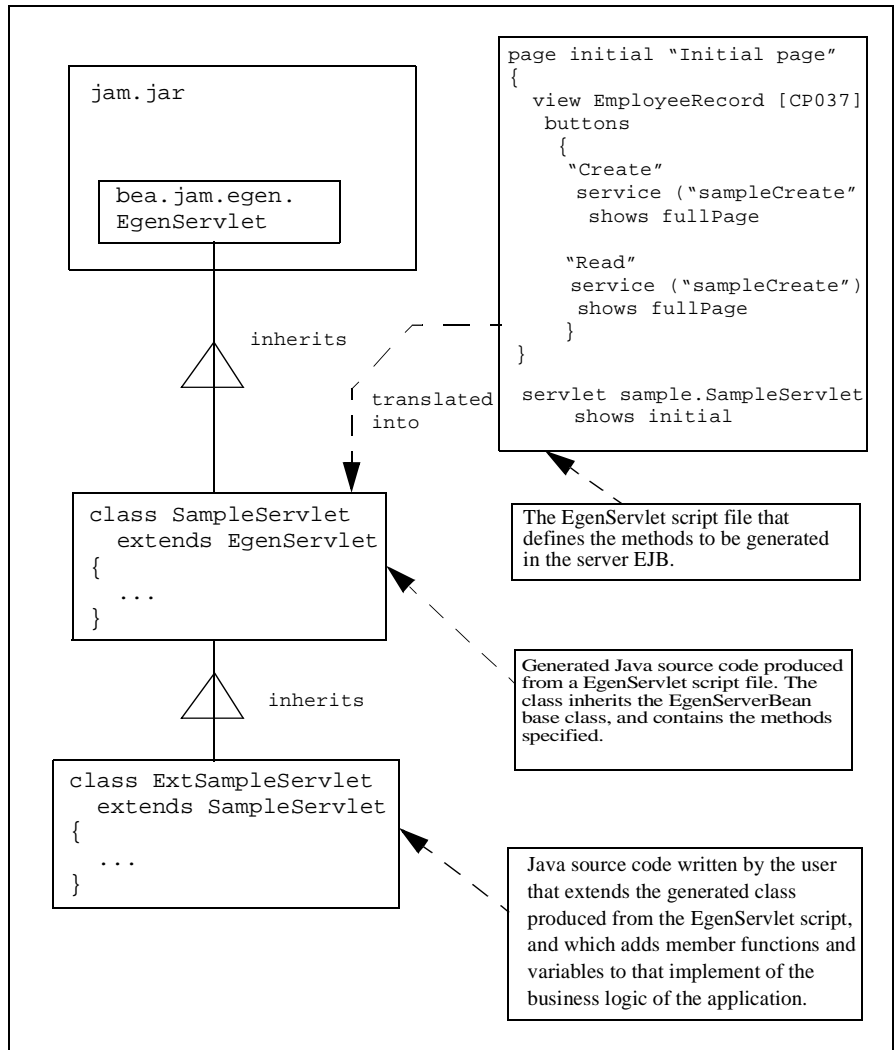
Customizing a Servlet-Only JAM Application

The generated Java classes produced for servlet applications are intended for proof of concept and prototypes, and can be customized in limited ways. It is presumed that some other development tool will be used to develop a servlet or other user interface on top of the generated EJB's or client classes.

This section describes the way that generated servlet code can be customized.

The following figure illustrates the relationships and inheritance hierarchy between the JAM classes comprising the application.

Figure 3-3 The JAM Servlet Class Hierarchy



The generated Java code for a servlet application is a class that inherits class `EgenServlet`. Class `EgenServlet` is provided in the JAM distribution jar file.

The base class illustrated in the following listing provides the basic framework for a servlet.

Listing 3-11 EgenServlet.java Base Class

```
//=====
// EgenServlet.java
//     The base class for generated servlets.
//=====

package bea.jam.egen;

//Imports

...

/*****
 * The base class for generated servlets
 */

abstract public class EgenServlet
    extends HttpServlet
{
    /** Perform an HTTP Get operation. */
    public void doGet(HttpServletRequest req,HttpServletResponse
resp)
        throws ServletException, IOException
    {
        DataView dv;
        HttpSession session=req.getSession(true);

        ...

        // Get the initial DataView data record
        dv = initialDataView();

        // Invoke the user-defined callback
        dv = doGetSetup(dv,session);

        // Convert the DataView into an HTML form
        ...
    }

    /** Perform a HTTP Post operation. */
    public void doPost(HttpServletRequest req,HttpServletResponse
resp)
        throws ServletException, IOException
    {
```

```
        DataView dv;
        HttpSession session=reqgetSession(true);

        // Move the HTML form data into a DataView
        ...

        // Invoke the user-defined callback
        dv = doPostSetup(dv, session);

        // Execute the form button
        ...

        //Invoke the user-defined callback
        dv = doPostFinal(dv, session);

        // Convert the DataView into an HTML form
        ...
    }

    /** User exit for pre-presentation processing for a GET request.
    */
    public DataView doGetSetup (DataView in, HttpSession session)
    {
        // Default behavior may be overridden
        return in;
    }

    /**User exit for before business logic processing for a POST
    request. */
    public DataView doPostSetup (DataView in, HttpSession session)
    {
        // Default behavior, may be overridden
        return in;
    }

    /** User exit for after business logic processing for a POST
    request. */
    public DataView doPostFinal (DataView in, HttpSession session)
    {
        // Default behavior, may be overridden
        return in;
    }

    /** Get an instance of the initial DataView for this servlet. */
    protected abstract DataView initialDataView();

    /**
     * The title for the initial page.
     * This should be initialized in the subclass constructor.
     */
    protected String m_initialTitle;
```

```
/**
 * The buttons for the initial page.
 * This should be initialized in the subclass constructor.
 */
protected Button[] m_initialButtons;
}

// End EgenServlet.java
```

The `EgenServlet` base class provides functions for the GET and POST operations for the servlet's HTML page.

Both of these operations invoke the following default callback functions:

- `doGetSetup()` - invoked before the GET operation.

This function occurs prior to the presentation of the HTML page to the user's browser. Any changes made to the `DataView` object will be reflected in the contents of the HTML page.

- `doPostSetup()` - invoked before the POST operation.

This function occurs after the HTML page is presented and the user activates a form button. The `DataView` is sent to the `doPostSetup()` function, which operates on its contents. For example, validating the contents of the fields.

- `doPostFinal()` - invoked after the POST operation.

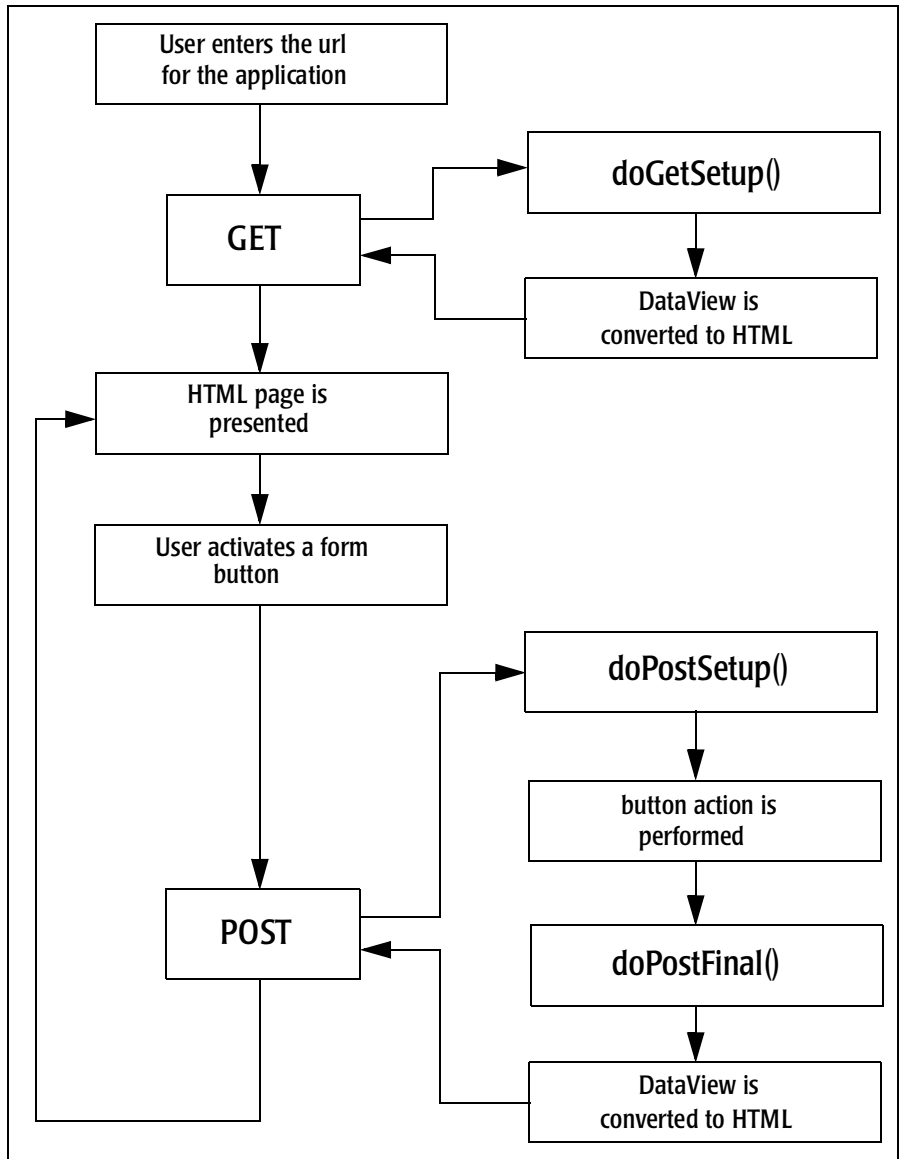
This function occurs prior to the presentation of the HTML page to the user's browser after activating a form button. Any changes made to the `DataView` object will be reflected in the contents of the HTML page.

Your class, which (indirectly) extends the `EgenServlet` base class, overrides these functions and provides additional business logic to modify the contents of the `DataView`. Each of these functions is passed to the `DataView` object containing the current record data. Each is expected to return a (potentially modified) `DataView` object.

Note: The overriding functions must have exactly the same signature as the functions in the base class.

The following illustration shows the sequence of operations that occur during the course of a user's browser session. For example, the series of events that occur within the `EgenServlet` class.

Figure 3-4 User Browser-Session Flowchart



Example ExtSampleServlet.java Class

The following listing shows an sample ExtSampleServlet class that extends the generated SampleServlet class, and adds a validation function (isSsnValid()) for the emp-ssn (m_empSsn) field of the DataView EmployeeRecord class. The three callback functions are overridden by the functions in the extended class. If the emp-ssn field is determined to be invalid, an exception is thrown.

Exceptions are caught by the Java server (WLS) and cause a simple informational text page to be presented to the user's browser. Any string text associated with the exception is displayed along with an execution stack trace that was in effect at the time that the exception was thrown.

Listing 3-12 Sample ExtSampleServlet.java Contents

```
//=====
// ExtSampleServlet.java
// Example class that extends a generated JAM servlet application.
//=====

package sample;

// System imports

import java.math.BigDecimal;

import javax.servlet.http.HttpSession;
import bea.dmd.DataView.DataView;
import bea.jam.egen.EgenServlet;

// Local Imports

import sample.EmployeeRecord;
import sample.SampleServlet;

/*****
 * Extends the SampleServlet class, adding additional business
 * logic
 */

public class ExtSampleServlet
    extends SampleServlet
{
    // Public functions
```

```

/*****
 * User exit for pre-presentation processing for a GET
 * request. This is called prior to the presentation of the
 * first HTML page to the user's browser.
 */

public DataView doGetSetup (DataView in HttpSession session)
{
    EmployeeRecord ereco;

    // Overrides the default behavior

    // Load default data into the empty DataView
    ereco = (EmployeeRecord) in;
    ereco.getEmpRecord().setEmpSsn(BigDecimal.valueOf(99999));

    return (ereco);
}

/*****
 * User exit for before business logic processing for a POST
 * request. This is called after the user activates a button
 * on the HTML form, but before the action associated with the
 * button is performed.
 */

public DataView doPostSetup (DataView in HttpSession session)
{
    EmployeeRecord ereco;

    // Overrides the default behavior

    // validate the Social Security Number field
    ereco = (EmployeeRecord) in;

    if (!isSsnValid(ereco.getEmpRecord().getEmpSsn()))
    {
        // The SSN is not valid
        throw new Error ("Invalid Social Security Number:"
            + ereco.getEmpRecord().getEmpSsn());
    }

    return (ereco);
}

/*****
 * User exit for after business logic processing for a POST
 * request. This is called after the action is performed for
 * the button on the HTML form is activated by the user.
 */

```

```
public DataView doPostFinal(DataView in HttpSession session)
{
    // Overrides the default behavior

    // Nothing to do here

    return (in);
}

// Private functions

/*****
 * Validates an SSN field.
 *
 * @return
 * True if the SSN is valid, otherwise false.
 */

private boolean isSsnValid(final BigDecimal ssn)
{
    if (ssn.longValue() < 100000000)
    {
        // Oops, the SSN should not have a leading zero
        return (false);
    }
    else
        return (true);
}

}

//End ExtSampleServlet.java
```

Once the `ExtSampleServlet` class has been written, it and the other servlet Java source files must be compiled and deployed in the same manner as other servlets.

Generating a Client Enterprise Java Bean-Based Application

This type of application produces Java classes that comprise an EJB application. The class methods are invoked from requests originating from other EJB classes, and transfer data records to and from the mainframe (remote system). From the viewpoint of the mainframe, the Java classes act as a remote DTP or IMS client. From the viewpoint of the EJB classes, they act as regular EJB classes.

Generating a Client Enterprise Java Bean-based application consists of the following steps;

1. Creating a script.
2. Processing a script.
3. Working with the generated files.
4. Customizing the application.

The following topics discuss these steps and offer examples of each.

Creating a Script

In order to produce an EJB-based application, the script file that defines your DataViews must be enhanced to describe both the mainframe services accessed and the EJB that will access them. A service description might look like the following listing example.

Listing 3-13 Sample service Description

```
service getSalary  
  
    accepts EmployeeRecord  
    returns EmployeeRecord
```

This sample listing defines a service named `getSalary` that accepts an input buffer of type `EmployeeRecord` and returns an output buffer of type `EmpDataView`. It is this service name which also requires an entry in the `jcrmgw.cfg` file.

An EJB that uses this service might be defined like the following listing.

Listing 3-14 Sample `getSalary` Service Definition

```
client ejb MyEJBNameMyEJBHome
{
    method getPay is service getSalary
}
```

This listing defines a Java bean class named `MyEJBName` with a method named `getPay`. The method corresponds to service name `getSalary`. The EJB home will be registered in JNDI under the name `MyEJBHome`.

The following listing shows the contents of a complete script file for defining a client EJB application.

Listing 3-15 Sample Client EJB Script

```
1  #-----
2  # empclient.egen
3  #   JAM script for an employee record.
4  #
5  #   $Id: empclient.egen,v 1.1 2000/01/25  18:34:14 david Exp$
6  #-----
7
8  # DataViews (typed data records)
9
10 view sample.EmployeeRecord (Comment 1)
11     from emprec.cpy
12
13 # Services
14
15 service sampleCreate (Comment 2)
16     accepts EmployeeRecord
17     returns EmployeeRecord
18
19 service sampleRead (Comment 2)
```

```
20     accepts EmployeeRecord
21     returns EmployeeRecord
22
23 service sampleUpdate (Comment 2)
24     accepts EmployeeRecord
25     returns EmployeeRecord
26
27 service sampleDelete (Comment 2)
28     accepts EmployeeRecord
29     returns EmployeeRecord
30
31 # Clients and servers
32
33 client ejb sample.SampleEBean my.sampleBean (Comment 3)
34 {
35     method newEmployee (Comment 4)
36         is service sampleCreate
37
38     method readEmployee (Comment 4)
39         is service sampleReads
40 }
41
42 # End
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-6 Script Comments

Comment 1	Defines a DataView class, specifying its corresponding copybook source file and its package name.
Comment 2	Defines a service function and its input and output parameter types.
Comment 3	Defines a client EJB class and its home name.
Comment 4	Defines a client class method and its service name.

Processing the Script

Issuing the following command will process the script.

Listing 3-16 Sample Script Process Command

```
% CLASSPATH=jam.jar java bea.jam.egen.EgenCobol empclient.egen  
emprec.cpy, Lines: 21, Errors: 0, Warnings: 0  
Generating sample.EmployeeRecord...  
Generating group emp-name  
Generating group emp-addr  
Generating SampleEBean...
```

Generated Files

This script command generates the following files.

Table 3-7 Sample Script Generated Files

File	Content
SampleEBean.java	Source for the EJB remote interface.
SampleEBeanBean.java	Source for the EJB implementation.
SampleEBeanHome.java	Source for the EJB home interface.
SampleEBeanDD.txt	Sample deployment descriptor.
EmployeeRecord.java	Source for the DataView object.

SampleEBean.java Source File

The following listing shows the contents of the generated `SampleEBean.java` source file.

Listing 3-17 Sample SampleEBean.java Contents

```
// SampleEBean.java
//
// EJB Remote Interface generated by EgenCobol on 24-Jan-2000.

package sample;

// Imports

import javax.ejb.EJBObject;
...

/** Remote Interface for SampleEBean EJB. */
public interface SampleEBean (Comment 1)
    extends EJBObject
{
    // newEmployee (Comment 2)
    EmployeeRecord newEmployee (EmployeeRecord commarea)
        throws RemoteException, UnexpectedException;

    readEmployee (Comment 2)
    EmployeeRecord readEmployee (EmploymentRecord commarea)
        throws RemoteException, UnexpectedException;
}

// End SampleEBean.java
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-8 Script Comments

Comment 1	Defines an EJB interface.
Comment 2	Methods to convert a raw COMMAREA into a Java DataView object.

SampleEBeanBean.java Source File

The following listing shows the contents of the generated SampleEBeanBean.java source file.

Listing 3-18 SampleEBeanBean.java Contents

```
// SampleEBeanBean.java
//
// EJB generated by EgenCobol on 24-Jan-2000.

package sample;

//Imports

import bea.jam.egen.EgenClientBean;
...

/** EJB implementation. */
public class SampleEBeanBean (Comment 1)
    extends EgenClientBean
{
    // newEmployee

    public EmployeeRecord newEmployee (EmployeeRecord commarea)
        throws IOException, snaException (Comment 2)
    {
        ...
    }

    //readEmployee

    public EmployeeRecord readEmployee (EmployeeRecord commarea)
        throws IOException, snaException (Comment 2)
    {
        ...
    }
}

// End SampleEBeanBean.java
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-9 Script Comments

Comment 1	Defines an EJB client bean.
-----------	-----------------------------

Table 3-9 Script Comments

Comment 2	The methods convert a raw COMMAREA into a Java DataView object.
-----------	---

SampleEBeanHome.java Source File

The following listing shows the contents of the generated `SampleEBeanHome.java` source file.

Listing 3-19 Sample `SampleEBeanHome.java` Contents

```
// SampleEBeanHome.java
//
// EJB Home interface generated by EgenCobol on 24-Jan-2000.

package sample;

// Imports

import javax.ejb.EJBHome;
...

/** Home interface for SampleEBean EJB. */
public interface SampleEBeanHome (Comment 1)
    extends EJBHome
{
    // create

    SampleEBean create()
        throws CreateException, RemoteException;
}

// End SampleEBeanHome.java
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-10 Script Comments

Comment 1	Defines an EJB home interface.
-----------	--------------------------------

SampleEBeanDD.txt Source File

The following listing shows the contents of the generated SampleEBeanDD.txt source file

Listing 3-20 Sample SampleEBeanDD.txt Contents

```
(SessionDescriptor
    beanHomename                my.sampleBean (Comment 1)
    enterprisebeanClassName      SampleEBeanBean
    homeInterfaceClassName       SampleEBeanHome
    remoteinterfaceclassName     SampleEBean
    isReentrant                  false
    ; session EJBBean-specific properties
    stateManagementType          STATELESS_SESSION
    sessionTimeout               5; seconds
    ; end session EJBBean-specific properties

    (accessControlEntries
;    DEFAULT
    ); end accessControlEntries

    (control Descriptors
        (DEFAULT
            isolationLevel        TRANSACTION_SERIALIZABLE
            transactionAttribute   TX_NOT_SUPPORTED
            runAsMode              CLIENT_IDENTITY
;            runAsIdentity        traderAdmin
        ); end DEFAULT
    ); end controlDescriptors

    (environmentProperties
        maxbeansInFreePool        100
        maxBeansInCache           100
        idleTimeoutSeconds        5
    ); end environmentProperties
); end SessionDescriptor
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-11 Script Comments

Comment 1	Defines an EJB session descriptor.
-----------	------------------------------------

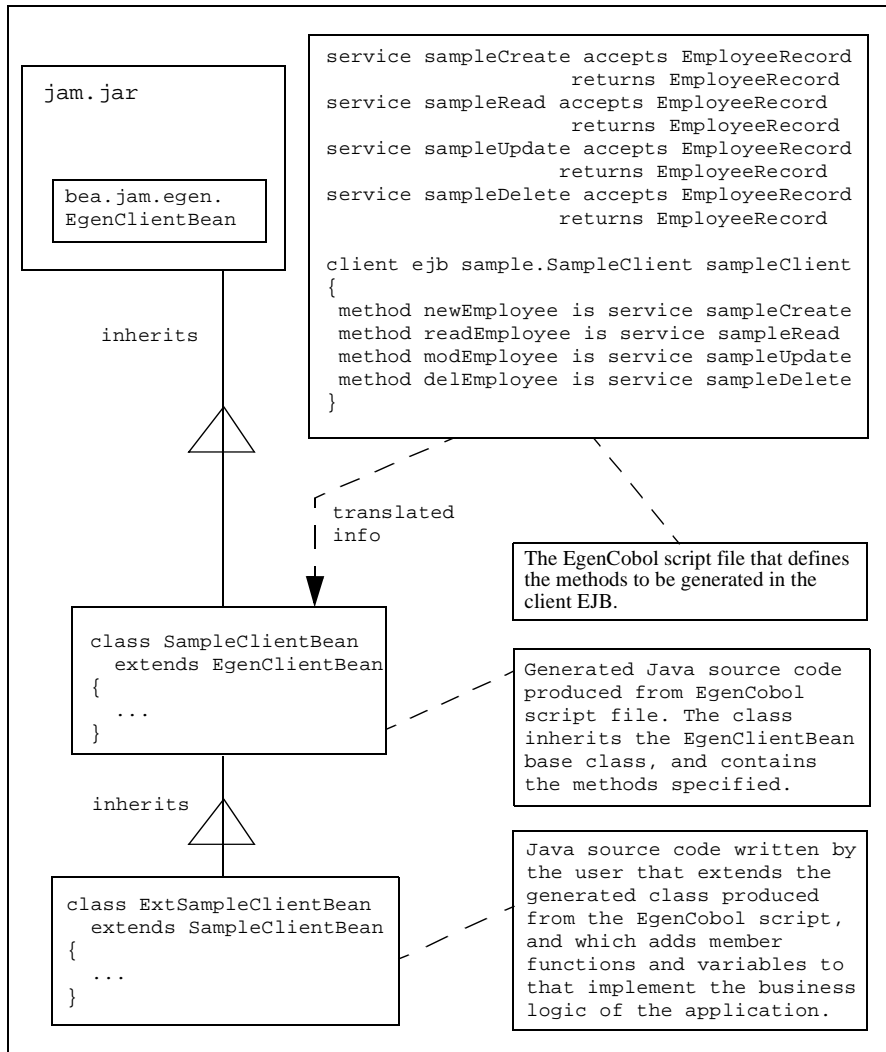
Customizing an Enterprise Java Bean-Based Application

Unlike the servlet applications, the generated Java classes produced for EJB applications are intended for customizing.

This section describes the way that generated client EJB code can be customized.

The following figure illustrates the relationships and inheritance hierarchy between the JAM classes comprising the application.

Figure 3-5 The JAM Client EJB Class Hierarchy



The generated Java code for a client EJB application is a class that inherits class `EgenClientBean`. The `EgenClientBean` class is provided in the JAM distribution jar file.

This base class, illustrated in the following listing, provides the basic framework for an EJB. It provides the required methods for a Stateless Session EJB.

Listing 3-21 EgenClientBean.java **Base Class**

```
//=====
// EgenClientBean.java
//   The base class for generated client EJB's.
//
//-----

package bea.jam.egen;

abstract public class EgenClientBean
    implements SessionBean
{
    //Implementation of ejbActivate(), ejbRemove(),
    // ejbPassivate(), ejbCreate() and setSessionContext()
    ...

    /**
     * Call a service by name through the jcrmgw.
     *
     * @exception bea.sna.jcrmgw.snaException For Gateway errors
     * @exception java.io.IOException For data translation
     *         errors.
     */
    protected byte[] callService(String service, byte[] in)
        throws snaException, IOException
    {
        // Low level gateway access code
        ...
    }

    // Variables

    protected transient SessionContext m_context;
    protected transient Properties      m_properties;
}

// End EgenClientBean.java
```

The generated class, illustrated in the following Listing, adds the methods specific to this EJB.

Listing 3-22 Generated SampleClientBean.java Class

```
// SampleClientBean.java
//
// EJB generated by EgenCobol on Feb 2, 2000.
//

package sample;

...

/**
 * EJB implementation.
 */
public class SampleClientBean extends EgenClientBean
{
    // deleteEmployee
    //
    public EmployeeRecord deleteEmployee(EmployeeRecord
commarea)
    throws IOException, snaException
    {
        // Make the remote call.
        //
        ...
    }

    //updateEmployee
    //
    public EmployeeRecord updateEmployee (EmployeeRecord
commarea)
        throws IOException, snaException
    {
        // Make the remote call.
        //
        ...
    }

    // readEmployee
    //
    public EmployeeRecord readEmployee (EmployeeRecord commarea)
        throws IOException, snaException
    {
        // Make the remote call.
        //
        ...
    }
}
```

```
// newEmployee
//
public EmployeeRecord newEmployee (EmployeeRecord commarea)
    throws IOException, snaException
{
    // Make the remote call.
    //
    ...
}

// END SampleClientBean.java
```

The following listing illustrates an example `ExtSampleClientBean` class that extends the generated `SampleClientBean` class, adding a validation function (`isSsnValid()`) for the `emp-ssn (m_empSsn)` field of the `DataView EmployeeRecord` class. The four methods are overridden by the methods in the extended class. If the `emp-ssn` field is determined to be invalid, an exception is thrown. Otherwise, the original function is called to perform the mainframe operation.

Listing 3-23 Example `ExtSampleClientBean.java` Class

```
// ExtSampleClientBean.java
//

package sample;

// Imports
//
import java.io.IOException;
import bea.jam.egen.EgenClientBean;
import bea.sna.jcrgw.snaException;
import bea.base.io.MainframeWriter;
import bea.base.io.MainframeReader;

/**
 * EJB implementation.
 */
public class ExtSampleClientBean extends SampleClientBean
{
    // deleteEmployee
    //
    public EmployeeRecord deleteEmployee (EmployeeRecord
    commarea)
```

```
        throws IOException, snaException
    {
        EmployeeRecord ereco = (EmployeeRecord) in;
        if (!isSsnValid (ereco.getEmpRecord().getEmpSsn()))
        {
            // The SSN is not valid
            throw new Error ("Invalid Social Security Number:" +
                            ereco.getEmpRecord().getEmpSsn());
        }

        // Make the remote call.
        //
        return super.deleteEmployee (commarea);
    }

    // updateEmployee
    //
    public EmployeeRecord updateEmployee (EmployeeRecord
        commarea)
        throws IOException, snaException
    {
        EmployeeRecord ereco = (EmployeeRecord) in;
        if (!isSsnValid (ereco.getEmpRecord().getEmpSsn()))
        {
            // The SSN is not valid.
            throw new Error("Invalid Social Security Number: "+
                            ereco.getEmpRecord().getEmpSsn());
        }

        // Make the remote call.
        //
        return super.updateEmployee (commarea);
    }

    // readEmployee
    //
    public EmployeeRecord readEmployee (EmployeeRecord commarea)
        throws IOException, snaException
    {
        EmployeeRecord ereco = (EmployeeRecord) in;
        if (!isSsnValid(ereco.getEmpRecord().getEmpSsn()))
        {
            // The SSN is not valid.
            throw new Error ("Invalid Social Security Number: "+
                            ereco.getEmpRecord().getEmpSsn());
        }

        // Make the remote call.
        //
    }
```

```
        return super.readEmployee (commarea);
    }

    //newEmployee
    //
    return super.readEmployee (commarea);
}

// new Employee
//
public EmployeeRecord newEmployee(EmployeeRecord commarea)
    throws IOException, snaException
{
    EmployeeRecord ereco = (EmployeeRecord) in;
    if (!isSsnValid(ereco.getEmpRecord().getEmpSsn()))
    {

        // The SSN is not valid.
        throw new Error("Invalid Social Security Number:"+
            ereco.getEmpRecord().getEmpSsn());
    }

    // Make the remote call.
    //
    return super.newEmployee(commarea);
}

// Private Functions

/*****
 * Validates an SSN field.
 */

private boolean isSsnValid(Big Decimal ssn)
{
    if (ssn.longValue() < 1000000000)
    {
        // Oops, should not have a leading zero.
        return false;
    }
    return true;
}

}

// END ExtSampleClientBean.java
```

Once the `ExtSampleClientBean` class has been written, it and the other EJB Java source files must be compiled and deployed in the same manner as other EJB's. Prior to deploying, the deployment descriptor must be modified; the `enterpriseBeanClassName` property must be set to the name of your extended EJB implementation class.

Generating a Server Enterprise Java Bean-Based Application

This type of application produces Java classes that comprise an EJB application, similar to a Client EJB application, but acting as a remote server from the viewpoint of the mainframe. The classes process service requests originating from the mainframe (remote) system, known as “inbound” requests, and transfer data records to and from the mainframe. From the viewpoint of the Java classes, they receive EJB method requests. From the viewpoint of the mainframe application, it invokes remote DPL or IMS programs.

Generating a Server Enterprise Java Bean-based application consists of the following steps:

1. Creating a script.
2. Processing the script.
3. Working with the generated files.
4. Customizing the application.

The following topics discuss these steps and offer examples of each.

Creating a Script

The following listing shows the contents of a complete script for defining a server EJB application.

Listing 3-24 Sample Server EJB Script

```
1  #-----
2  # empserver.egen
3  #   JAM script for an employee record.
4  #
5  # $Id: empserver.egen, v 1.1 2000/01/21 23:20:40
6  #-----
7
8  # DataViews (typed data records)
9
10 view sample.EmployeeRecord (Comment 1)
11     from emprec.cpy
12
13 # Clients and servers (Comment 2)
14
15 server ejb sample.SampleEEServer my.sampleServer (Comment 3)
16 {
17     method newEmployee (EmployeeRecord) (Comment 4)
18         returns EmployeeRecord
19 }
20
21 # End
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-12

Comment 1	Defines a DataView class, specifying its corresponding copybook source file and its package name.
Comment 2	Defines a server EJB class.

Table 3-12

Comment 3	my.sampleServer is the home interface identifier for this bean. This value must be included in an entry in the local Services section of the jcrmgw.cfg file for the Java gateway.
Comment \$	Defines a server class method and its parameter.

Processing the Script

Issuing the following command will process the script.

Listing 3-25 Sample Script Process Command

```
% CLASSPATH=jam.jar java bea.jam.egen.EgenCobol empserver.egen
emprec.cpy, Lines: 21, Errors: 0, Warnings: 0
Generating sample.EmployeeRecord...
Generating group emp-name
Generating group emp-addr
Generating SampleEServer...
```

Generated Files

This script command generates the following files.

Table 3-13 Sample Script Generated Files

File	Content
SampleEServer.java	Source for the EJB remote interface.
SampleEServerBean.java	Source for the EJB implementation.
SampleEServerHome.java	Source for the EJB jhome interface.
SampleEServerDD.txt	Sample deployment descriptor.
EmployeeRecord.java	Source for the DataView object.

SampleEServer.java Source File

The following listing shows the content of the generated `SampleEServer.java` source file.

Listing 3-26 Sample `SampleEServer.java` Contents

```
// SampleEServer.java
//
//  EJB Remote Interface generated by EgenCobol on 24-Jan-2000.

package sample;

// Imports

import javax.ejb.EJBObject;
...

/** Remote Interface for SampleEServer EJB. */

public interface SampleEServer
    extends gwObject
{
    //dispatch
    byte[] dispatch(byte[] commarea, Object future)
        throws RemoteException, UnexpectedException;
}

// End SampleEServer.java
```

SampleEServerBean.java Source File

The following listing shows the contents of the generated `SampleEServerBean.java` source file.

Listing 3-27 Sample `SampleEServerBean.java` Contents

```
// SampleEServerBean.java
//
//  EJB generated by EgenCobol on 24-Jan-2000.

package sample;
```

```
// Imports

import bea.jam.egen.EgenServerBean;
...

/** EJB implementation. */

public class SampleEServerBean
    extends EgenServerBean
{
    // dispatch
    public byte[] dispatch (byte[] commarea, Object future)
        throws IOException
    {
        ...
    }

    /**
     * Do the actual work for a newEmployee operation.
     * NOTE: This routine should be overridden to do actual work
     */
    EmployeeRecord newEmployee (EmployeeRecord commarea)
    {
        return new EmployeeRecord();
    }
}

//End SampleEServerBean.java
```

SampleEServerHome.java Source File

The following listing shows the contents of the generated SampleEServerHome.java source file.

Listing 3-28 Sample SampleEServerHome.java Contents

```
// SampleEServerHome.java
//
// EJB Home interface generated by EgenCobol on 24-Jan-2000.

package sample;

//Imports
```

```
import javax.ejb.EJBHome;
...

/** Home interface for SampleEServer EJB. */

public interface SampleEServerHome
    extends EJBHome
{
    //create
    SampleEServer create()
        throws CreateException, RemoteException;
}

// End SampleEServerHome.java
```

SampleEServerDD.txt Source File

The following listing shows the contents of the generated SampleEServerDD.txt source file.

Listing 3-29 Sample SampleEServerDD.txt Contents

```
(SessionDescriptor
  beanHomeName                StatelessSession.my.sampleServer
  enterprisebeanClassName     SampleEServerBean
  homeInterfaceClassName      SampleEServerHome
  remoteInterfaceClassName    SampleEServer (Comment 1)
  isReentrant                  false
  ; session EJBBean-specific properties
  stateManagementType        STATELESS_SESSION
  sessionTimeout              5; seconds
  ; end session EJBBean-specific properties

  (accessControlEntries
  ;    DEFAULT
  ); end accessControlEntries

  (controlDescriptors
    DEFAULT
      isolationLevel            TRANSACTION_SERIALIZABLE
      transactionAttribute      TX_NOT_SUPPORTED
      runAsMode                  CLIENT_IDENTITY
  ;    runAsIdentity            traderAdmin
  ); end DEFAULT
  ); end controlDescriptors
```

```
(environmentProperties
    maxBeansInFreePool      100
    maxBeansInCache         100
    idleTimeoutSeconds      5
); end environmentProperties
); end SessionDescriptor
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-14 Script Comments

Comment 1	Defines an EJB session descriptor.
-----------	------------------------------------

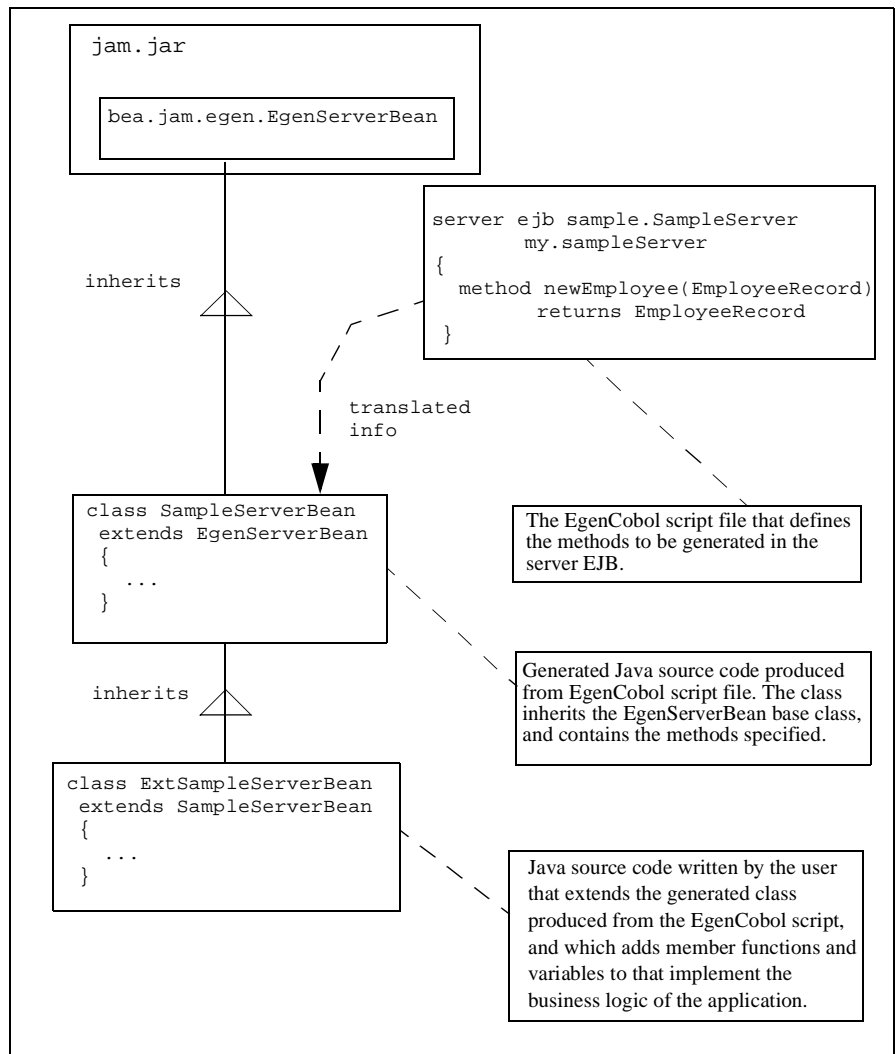
Customizing a Server Enterprise Java Bean-Based Application

The generated server enterprise Java bean-based applications are only intended for customizing, since they perform no real work without customizing.

This section describes the way that generated server EJB code can be customized.

The following figure illustrates the relationships and inheritance hierarchy between the JAM classes comprising the application.

Figure 3-6 The JAM Server EJB Class Hierarchy



The generated Java code for a client EJB application is a class that inherits class `EgenServerBean`. The `EgenServerBean` class is provided in the JAM distribution jar file. This base class, illustrated in the following listing, provides the basic framework for an EJB. It provides the required methods for a Stateless Session EJB.

Listing 3-30 EgenServerBean.java Base Class

```
//=====
// EgenServerBean.java
//     The base class for generated server EJB's.
//
//=====

package bea.jam.egen;

abstract public class EgenServerBean
    implements SessionBean
{
    // Implementation of ejbActivate(), ejbRemove(),
    // ejbPassivate(),
    // setSessionContext() and ejbCreate().
    ...

    // Variables

    protected transient SessionContext m_context;
    protected transient Properties m_properties;
}

// End EgenServerBean.java
```

The generated class, illustrated in the following listing, adds the methods specific to this EJB.

Listing 3-31 Generated SampleServerBean.java Class

```
// SampleServerBean.java
//
// EJB generated by EgenCobol on 03-Feb-00.
//

package sample;

//Imports
//
import java.io.IOException;
import java.util.Hashtable;
import bea.sna.jcrmgw.snaException;
import bea.base.io.MainframeWriter
```

```
import bea.base.io.MainframeReader;
import bea.jam.egen.EgenServerBean;
import bea.jam.egen.InboundDispatcher;

/**
 * EJB implementation
 */
public class SampleServerBean extends EgenServerBean
{
    // dispatch
    //
    public byte[] dispatch(byte[] commarea, Object future)
        throws IOException
    {
        EmployeeRecord    inputBuffer
                        = new EmployeeRecord (new
                                                MainframeReader (commarea));
        EmployeeRecord    result = newEmployee (inputBuffer);
        return result.toByteArray (new MainframeWriter());
    }

    /**
     * Do the actual work for a newEmployee operation.
     * NOTE: This routine should be overridden to do actual work
     */
    EmployeeRecord newEmployee(EmployeeRecord commarea)
    {
        return new EmployeeRecord();
    }
}

// END SampleServerBean.java
```

The following listing shows an example `ExtSampleServerBean` class that extends the generated `SampleServerBean` class, providing an implementation of the `newEmployee()` method. The example method merely prints a message indicating that a `newEmployee` request has been received.

Listing 3-32 Sample ExtSampleServerBean.java Contents

```
// ExtSampleServerBean.java
//

package sample;

/**
 * EJB implementation
 */
public class ExtSampleServerBean extends SampleServerBean
{
    public EmployeeRecord newEmployee (EmployeeRecord in)
    {
        System.out.println("New Employee: " +
            in.getEmpName().getEmpNameFirsr()
            + " " + in.getEmpname().getEmpNameLast());
        return in;
    }
}

// END ExtSampleServerBean.java
```

Once the `ExtSampleServerBean` class has been written, it and the other EJB Java source files must be compiled and deployed in the same manner as other EJB's. Before deploying, the deployment descriptor must be modified; the *enterpriseBeanClassName* must be set to the name of your extended EJB implementation class.

Generating a Stand-Alone Client Application

This type of application produces simple Java classes that perform the appropriate conversions of data records sent between Java and the mainframe, but without all of the EJB support methods. These classes are intended to be lower-level components upon which more complicated applications are built.

Generating a stand-alone application consists of the following steps;

1. Generating a script.
2. Processing a script.
3. Working with the generated files.
4. Customizing the application.

The following listing shows the contents of a complete script for defining a stand-alone client class with multiple services.

Listing 3-33 Sample Stand-Alone Client Class Script

```
1  #-----
2  # empclass.egen
3  #   JAM script for an employee record.
4  #
5  # $Id: empclass.egen, v 1.1 2000/01/21 23:20:40
6  #-----
7
8  # DataViews (typed data records)
9
10 view sample.EmployeeRecord (Comment 1)
11     from emprec.cpy
12
13     # Services
14
15     service sampleCreate (Comment 2)
16         accepts EmployeeRecord
17         returns EmployeeRecord
18
19     service sampleRead (Comment 2)
20         accepts EmployeeRecord
21         returns EmployeeRecord
22
23     service sampleUpdate (Comment 2)
24         accepts EmployeeRecord
25         returns EmployeeRecord
26
27     service sampleDelete (Comment 2)
28         accepts EmployeeRecord
29         returns EmployeeRecord
30
31     # Clients and servers
```

```

32
33  client class sample.SampleClass (Comment 3)
34  {
35      method newEmployee (Comment 4)
36          is service sampleCreate
37
38      method readEmployee (Comment 4)
39          is service sampleRead
40  }
41
42  # End

```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-15 Script Comments

Comment 1	Defines a DataView class, specifying its corresponding copybook source file and its package name.
Comment 2	Defines a service function and its input and output parameter types.
Comment 3	Defines a simple client class.
Comment 4	Defines a client class method and its parameter types.

Processing a Script

Issuing the following command will process the script.

Listing 3-34 Sample Script Process Command

```

% CLASSPATH=jam.jar java bea.jam.egen.EgenCobol empclass.egen
emprec.cpy, Lines: 21, Errors: 0, warnings: 0
Generating sample.EmployeeRecord...
Generating group emp-name
Generating group emp-addr
Generating SampleClass...

```

Generated Files

This script command generates the following files.

Table 3-16 Sample Script Generated Files

File	Content
SampleClass.java	Source for the sample class.
EmployeeRecord.java	Source for the DataView class.

SampleClass.java Source File

The following listing contains the generated SampleClass.java source file.

Listing 3-35 Sample SampleClass.java Source File

```
// SampleClass.java
//
// Client class generated by EgenCobol on 24-Jan-2000.

package sample;(Comment 1)

// Imports

import bea.jam.egen.EgenClient;
...

/* Mainframe client class. */

public class SampleClass (Comment 2)
    extends EgenClient
{
    // newEmployee
    public EmployeeRecord newEmployee (EmployeeRecord commarea)
        throws IOException, snaException (Comment 3)
    {
        ...
    }

    // readEmployee
    public EmployeeRecord readEmployee (EmployeeRecord commarea)
        throws IOException, snaException (Comment 3)
    {
        ...
    }
}
```

```
    }  
}  
  
// End SampleClass.java
```

Script Comments

The following numbered comments refer to the numbered comments in the prior listing.

Table 3-17 Script Comments

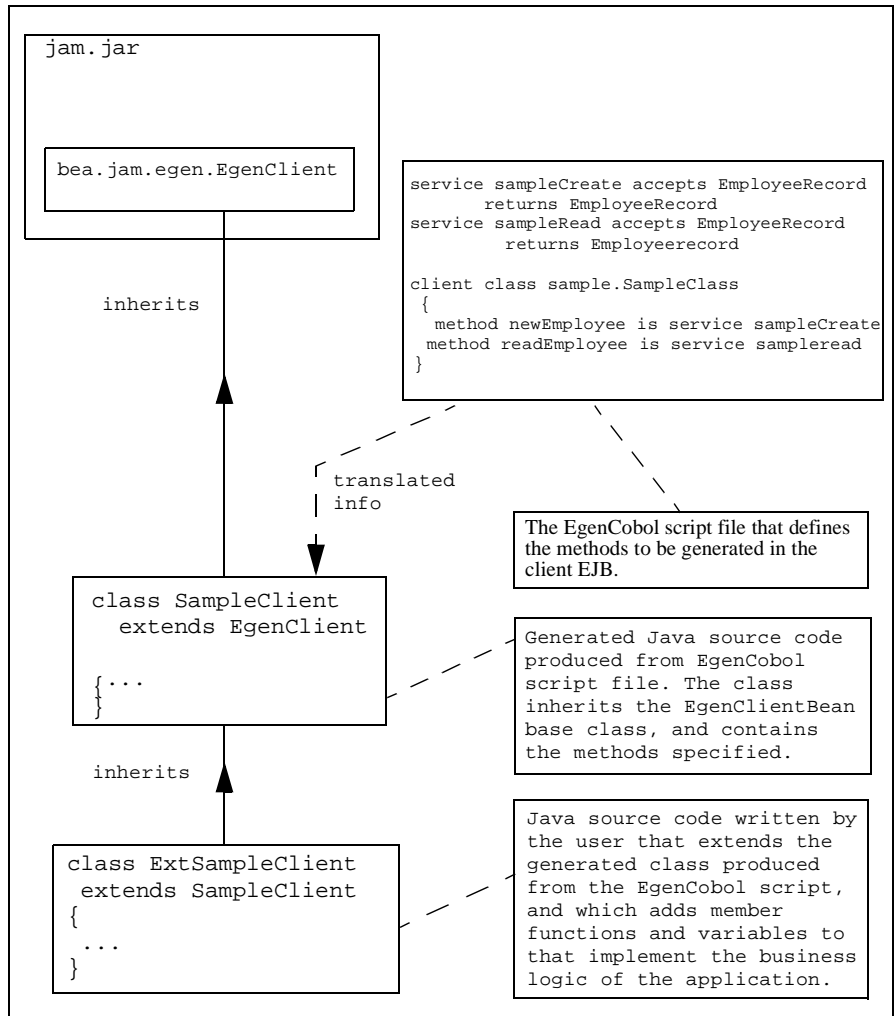
Comment 1	The package name is defined in the Egen script.
Comment 2	The data record is encapsulated in a class that extends the <code>EgenClient</code> class.
Comment 3	The methods convert a raw <code>COMMAREA</code> into a Java <code>DataView</code> object.

Customizing a Stand-Alone Java Application

The stand-alone client class model is the simplest JAM code generation model both in terms of the code generated and customizing the generated code.

The following figure illustrates the relationships and inheritance hierarchy between the JAM classes comprising the application.

Figure 3-7 The JAM Client EJB Class Hierarchy



The generated Java code for a client class application is a class that inherits class `EgenClient`. The `EgenClient` class is provided in the JAM distribution jar file. This base class, illustrated in the following listing provides the basic framework for a client to the `jcrmgw`. It provides the required methods for accessing the gateway.

Listing 3-36 Generated EgenClient.java Class

```
//=====
// EgenClient.java
//      Basic functionality for clients of the jcrmgw
//
//-----

package bea.jam.egen;

public class EgenClient
{
    public byte[] callService(String service, byte[] in)
        throws snaException, IOException
    {
        // make a mainframe request through the gateway.
        ...
    }
}

// End EgenClientBean.java
```

The generated class, illustrated in the following listing, adds the methods specific to the users application

Listing 3-37 Sample SampleClient.java Class

```
// SampleClass.java
//
// Client class generated by EgenCobol on 02-Feb-00.
//

package sample;

// Imports
//
import java.io.IOException;
import bea.jam.egen.EgenClient;
import bea.sna.jcrmgw.snaException;
import bea.base.io.MainframeWriter;
import bea.base.io.MainframeReader;
```

```
/**
 * Mainframe client class.
 */
public class SampleClass extends EgenClient
{
    // newEmployee
    //
    public EmployeeRecord newEmployee(EmployeeRecord commarea)
        throws IOException, SnaException
    {
        // Make the remote call.
        //
        byte[] inputBuffer = commarea.toByteArray(new
            MainframeWriter());
        byte[] rawResult = callService("sampleCreate",
            inputBuffer);
        EmployeeRecord result =
            new EmployeeRecord(new
                MainframeReader(rawResult));
        return result;
    }
    // readEmployee
    //
    public EmployeeRecord readEmployee(EmployeeRecord commarea)
        throws IOException, SnaException
    {
        // Make the remote call.
        //
        byte[] inputBuffer = commarea.toButeArray(new
            MainframeWriter());
        byte[] rawResult = callService("sampleRead", inputBuffer);
        EmployeeRecord result =
            new EmployeeRecord(new MainframeReader(rawResult));
        return result;
    }
}

// End SampleClass.java
```

Your class, which extends or uses the `SampleClient` class, simply overrides or calls these methods to provide additional business logic, modifying the contents of the `DataView`. It may also add additional methods, if desired.

The following listing shows an example `ExtSampleClass` class that extends the generated `SampleClient` class.

Listing 3-38 Sample ExtSampleClient.java Contents

```
// ExtSampleClient.java
//

package sample;

// Imports
//
import java.io.IOException;
import bea.jam.egen.EgenClientBean;
import bea.sna.jcrgw.snaException;
import bea.base.io.MainframeWriter;
import bea.base.io.MainframeReader;

/**
 * Extended Sample Class
 */
public class ExtSampleClient extends SampleClass
{
    // deleteEmployee
    //
    public EmployeeRecord deleteEmployee(EmployeeRecord
        commarea)
        throws IOException, snaException
    {
        EmployeeRecord ereco=(EmployeeRecord) in;
        if (!isSsnValid(ereco.getEmpRecord().getEmpSsn()))
        {
            // The SSN is not valid.
            throw new Error ("Invalid Social Security Number:"+
                ereco.getEmpRecord().getEmpSsn());
        }

        // Make the remote call.
        //
        return super.deleteEmployee(commarea);
    }

    //updateEmployee
    //
    public EmployeeRecord updateEmployee(EmployeeRecord
        commarea)
        throws IOException, snaException
    {
        EmployeeRecord ereco = (EmployeeRecord) in;
        if (!isSsnValid(ereco.getEmpRecord().getEmpSsn()))
        {
            The SSN is not valid.
        }
    }
}
```

```
        throw new Error ("Invalid Social Security Number:" +
            erec.getEmpRecord().getEmpSsn());
    }

    // Make the remote call.
    //
    return super.updateEmployee(commarea);
}

// readEmployee
//
public EmployeeRecord readEmployee(EmployeeRecord commarea)
    throws IOException, snaException
{
    EmployeeRecord erec = (EmployeeRecord)in;
    if (!isSsnValid(erec.getEmpRecord().getEmpSsn()))
    {
        // The SSN is not valid.
        throw new Error("Invalid Social Security Number:" +
            erec.getEmpRecord().getEmpSsn());
    }

    // Make the remote call.
    //
    return super.readEmployee(commarea);
}

//newEmployee
//
public EmployeeRecord newEmployee(EmployeeRecord commarea)
    throws IOException, snaException
{
    EmployeeRecord erec = (EmployeeRecord) in;
    if (!isSsnValid(erec.getEmpRecord().getEmpSsn()))
    {
        // The SSN is not valid.
        throw new Error("Invalid Social Security Number:" +
            erec.getEmpRecord().getEmpSsn());
    }

    // Make the remote call.
    //
    return super.newEmployee(commarea);
}

// Private functions

/*****
 * Validates an SSN field.
 */
```

```
        private boolean isSsnValid(BigDecimal ssn)
        {
            if (ssn.longValue() < 1000000000)
            {
                // Ops, should not have a leading zero.
                return false;
            }

            return (true);
        }
    ]

// END ExtSampleClient.java
```

Once the `ExtSampleClient` class has been written, it and the other Java source files must be compiled and placed on to your `WEBLOGICCLASSPATH`.

4 Deploying Applications

This chapter describes how to deploy your own applications using the BEA eLink Java Adapter for Mainframe WLS Edition (JAM) software. Deployment is the process of taking previously developed servlets and/or EJB's and installing them into a specific operational environment. In this case, the operational environment is your WebLogic system.

This topic consists of the following sub-topics:

- Deploying Servlets
- Deploying Enterprise Java Beans

Deploying Servlets

Please refer to the BEA WebLogic Server documentation for detailed instructions on deploying servlets.

The following steps provide an example of deploying a servlet by modifying the `weblogics.properties` file.

Perform the following steps to deploy a servlet:

1. Place a class file for the servlet and all other classes that it depends on into the servlet classpath. Be sure and include the class files for the `DataView`'s that are used by the servlet.
2. Add a registration line to your `weblogics.properties` file. This line will specify both the URL that will be used to access the servlet and the full package and name of the servlet class. For example, assuming your server is named `www.webstore.com`, the properties line `weblogic.httpd.register.widgets=webstore.WidgetServlet` will cause the class `webstore.WidgetServlet` to serve requests to the URL `http://www.webstore.com/widgets`.

```
weblogic.httpd.register.urlname=mypackage.MyServlet
```

This registration line causes the WebLogic Server to use the servlet class “`mypackage.MyServlet`” to serve all HTTP requests for a page at the URL name (on your host).

3. Add any corresponding remote service entries into the `jcrmgw.cfg` file. For example, if your servlet invokes a service named `widgetQuote`, then you would need something like the following in your configuration:

```
*JC_REMOTE_SERVICES
widgetQuote RDOM="myMainframe"
            RNAME="WIDQUOTE"
```

4. Restart the WebLogic Server.

There are other servlet deployment options, including hot-deploying a servlet into a running WebLogic server. Please refer to the WebLogic Server documentation for more details.

Deploying Enterprise Java Beans

Please refer to the BEA WebLogic Server documentation for detailed instructions on deploying EJBs.

Perform the following steps to deploy an EJB:

1. Place the class file for the EJB and all other classes that it depends on into your development classpath. Be sure to include the class files for the DataView's that are used by the EJB.
2. Convert the text deployment descriptor into serialized form with the following command.

```
java weblogic.ejb.utils.DDCreator MyEjbDD.txt
```

This command creates the serialized deployment descriptor in the current directory.

3. Build the Home Interface implementation, remote Interface implementation and container classes by using the following command.

```
java weblogic.ejbrc -d dest MyEjbDD.ser
```

Replace 'dest' with the directory where you want the generated class files to be placed.

4. Place the class files for the generated files and the EJB (including classes it depends on) into your WebLogic classpath. Be sure and include the class files for the DataView's that are used by the EJB.
5. Edit your Weblogic properties file to add the new EJB to the `weblogic.ejb.deploy` property. This property should contain a comma separated list of full-paths to ejb ".ser" or ".jar" files. For example:

```
weblogic.ejb.deploy=c:/mybeans/MyEjbDD.ser,c:/otherbeans/  
WebstoreEJB.jar
```

6. Add any corresponding local or remote service entries into the `jcrmgw.cfg` file. For example, if your EJB invokes a service named "widgetQuote", then you would need something like the following in your configuration:

```
*JC_REMOTE_SERVICES  
widgetQuote RDOM="myMainframe"  
            RNAME="WIDQUOTE"
```

7. Restart WLS.

EJB's may also be placed into jar files before being deployed, and may be hot-deployed into a running WebLogic Server. Please refer to the Weblogic Server documentation for more details.

5 Security

BEA eLink Java Adapter for Mainframe WLS Edition (JAM) supports the basic Application Program-to-Program Communication (APPC) style of sign-on security. You can configure a gateway to use one of three types of sign-on security for each link that is defined (refer to the “JC_SNALINKS Section” in “Configuring the Java Adapter for Mainframe Environment.”) The selected level of security determines what combination of userid and password is used for transactions across the link.

JAM supports the following security options:

- **LOCAL**
All security is handled by the local system and the link itself has no security requirement.
- **IDENTIFY**
A userid is passed to the mainframe. This userid can originate with the client application or it can be a default userid supplied at Java gateway startup by the `-u` option in the WebLogic Server (WLS) `properties` file as part of the `startupArgs` entry for the gateway.
- **VERIFY**
A userid and password are passed to the mainframe. The userid can originate with the client application or it can be a default userid supplied at Java gateway startup by the `-u` option in the WLS `properties` file as part of the `startupArgs` entry for the gateway. The password must be supplied by the client application.

In addition, an alternate mirror transaction is supported on each Distributed Program Link (DPL) which can be used to associate different Resource Access Control Facility (RACF) profiles with different services.

Refer to IBM RACF documentation for more specific information about establishing and administrating mainframe security.

6 Programming Reference

This section provides the rules that enable you to identify what form a generated Java class takes from a given COBOL copybook processed by the `EgenCobol` tool. This facilitates your being able to correctly code any custom programs that make use of the generated classes.

The `EgenCobol` tool maps a COBOL copybook into a Java class. The COBOL copybook contains a data record description. The `EgenCobol` tool derives the generated Java class from the `bea.dmd.dataview.DataView` class (later referred to as `DataView`), which is provided on your BEA eLink Java Adapter for Mainframe WLS Edition (JAM) product CDROM in the `jam.jar` file.

This section gives you data mapping rules. The following topics are discussed:

- Field Name Mapping Rules
- Field Type Mappings
- Group Field Accessors
- Elementary Field Accessors
- Array Field Accessors
- Fields with REDEFINES Clauses
- COBOL Data Types

You should find the COBOL terms in this section easy to understand; however, you may need to use a COBOL reference book or discuss the terms with a COBOL programmer. Also, you can process a copybook with the `EgenCobol` tool and examine the produced Java code in order to understand the mapping.

Field Name Mapping Rules

When you process a COBOL copybook containing field names, they are mapped to Java names. This is performed by the `EgenCobol` tool according to the following rules:

1. All alphabetic characters are mapped to lower case.
2. All dashes are removed and the character following the dash is mapped to upper case.
3. When a prefix is added to the name (as when creating a field accessor function name) the first character of the base name is mapped to upper case.

Table 6-1 lists some mapping examples.

Table 6-1 Example Field Name Mapping from COBOL to Java and Accessor

COBOL Field Name	Java base name	Sample Accessor Name
EMP-REC	empRec	setEmpRec
500-REC-CNT	500RecCnt	set500RecCnt

Field Type Mappings

When you process a COBOL copybook with field types, the field types are mapped to Java field types. This is performed by the `EgenCobol` tool according to the following rules:

1. Group fields map to `DataRow` subclasses.
2. All alphanumeric fields are mapped to type `String`.
3. All edited numeric fields are mapped to type `String`.
4. All `SIGN IS LEADING`, `SIGN IS TRAILING`, `BLANK WHEN ZERO` or `JUSTIFIED RIGHT` fields are mapped to type `String`.

5. The types COMP-1, COMP-2, COMP-5, COMP-X, POINTER and PROCEDURE-POINTER fields are not supported (an error message is generated).
6. All INDEX fields are mapped to Java type `int`.
7. All numeric fields with any digits to the right of the decimal point are mapped to type `BigDecimal`.
8. All COMP-3 (packed) fields are mapped to type `BigDecimal`.
9. All other numeric fields are mapped as shown in Table 6-2.

Table 6-2 Numeric Field Mapping

Number of digits	Java Type
≤ 4	<code>short</code>
> 4 and ≤ 9	<code>int</code>
> 9 and ≤ 18	<code>long</code>
> 18	<code>BigDecimal</code>

Group Field Accessors

Each nested group field in a COBOL copybook is mapped to a corresponding `DataView` subclass. The generated subclasses are nested exactly as the COBOL groups in the copybook. In addition, the `EgenCobol` tool generates a private instance variable of this class type and a `get` accessor.

For example, the following copybook:

```
10 MY-RECORD.
   20 MY-GRP.
      30 ALNUM-FIELD                PIC X(20).
```

Produces code similar to the following:

```
public MyGrpV getMyGrp();
public static class MyGrpV extends DataView
{
    // Class definition
}
```

Elementary Field Accessors

Each elementary field is mapped to a private instance variable within the generated DataView subclass. Access to this variable is accomplished by two accessors that are generated (set and get).

These accessors have the following forms:

```
public void setFieldName(FieldType value);
public FieldType getFieldName();
```

Where:

FieldType
is described in the “Field Type Mappings” section.

FieldName
is described in the “Field Name Mapping Rules” section.

For example, the following copybook:

```
10 MY-RECORD.
    20 NUMERIC-FIELD          PIC S9(5).
    20 ALNUM-FIELD            PIC X(20).
```

Produces the accessors:

```
public void setNumericField(int value);
public int getNumericField();
public void setAlnumField(String value);
public String getAlnumField();
```

Array Field Accessors

Array fields are handled according to the other rules in this section, with the addition that each accessor takes an additional `int` argument that specifies which array entry is to be accessed, for example:

```
public void          setFieldName(int index, FieldType value);
public FieldType    getFieldName(int index);
```

Array fields specified with the `DEPENDING ON` clause are handled the same as fixed-size arrays with the following special rules:

1. The accessors may be used to `get` or `set` any instance up to the maximum array index.
2. The controlling (`DEPENDING ON`) variable is evaluated when the `DataView` is converted to or from an external format, such as a mainframe format. The `EgenCobol` tool converts only the array elements with subscripts less than the controlling value.

Fields with REDEFINES Clauses

Fields that participate in a `REDEFINES` set are handled as a unit. A private `byte[]` variable is declared to hold the underlying mainframe data, as well as a private `DataView` variable. Each of the redefined fields has an accessor or accessors. These accessors take more CPU overhead than the normal accessors because they perform conversions to and from the underlying `byte[]` data.

For example the copybook:

```
10 MY-RECORD.
   20 INPUT-DATA.
       30 INPUT-A                PIC X(4).
       30 INPUT-B                PIC X(4).
   20 OUTPUT-DATA REDEFINES INPUT-DATA PIC X(8).
```

Produces Java code similar to the following:

```
private byte[] m_redef23;  
private DataView m_redef23DV;  
public InputDataV getInputData();  
public String getOutputData();  
public void setOutputData(String value);  
public static class InputDataV extends DataView  
{  
    // Class definition.  
}
```

COBOL Data Types

This section summarizes the COBOL data types supported by JAM software. Table 6-3 lists the COBOL data item definitions recognized by the EgenCobol tool. Table 6-4 lists the syntactical features and data types recognized by the EgenCobol tool. If a COBOL feature is unsupported, an error message is generated, unless it is listed as ignored in a table.

Table 6-3 Major COBOL Features

COBOL Feature	Support
IDENTIFICATION DIVISION	unsupported
ENVIRONMENT DIVISION	unsupported
DATA DIVISION	partially supported
WORKING-STORAGE SECTION	partially supported
<i>Data record definition</i>	supported
PROCEDURE DIVISION	unsupported
COPY	unsupported
COPY REPLACING	unsupported
EJECT, SKIP1, SKIP2, SKIP3	supported

Table 6-4 COBOL Data Types

COBOL Type	Java Type
COMP, COMP-4, BINARY (<i>integer</i>)	short/int/long
COMP, COMP-4, BINARY (<i>fixed</i>)	BigDecimal
COMP-3, PACKED-DECIMAL	BigDecimal
COMP-5	not supported
COMP-X	not supported
DISPLAY <i>numeric (zoned)</i>	BigDecimal
BLANK WHEN ZERO (<i>zoned</i>)	String
SIGN IS LEADING (<i>zoned</i>)	String
SIGN IS LEADING SEPARATE (<i>zoned</i>)	String
SIGN IS TRAILING (<i>zoned</i>)	String
SIGN IS TRAILING SEPARATE (<i>zoned</i>)	String
<i>edited numeric</i>	String
COMP-1, COMP-2 (<i>float</i>)	not supported
<i>edited float numeric</i>	String
DISPLAY (<i>alphanumeric</i>)	String
<i>edited alphanumeric</i>	String
INDEX	int
POINTER	not supported
PROCEDURE-POINTER	not supported
JUSTIFIED RIGHT	not supported (ignored)
SYNCHRONIZED	not supported (ignored)
REDEFINES	supported

Table 6-4 COBOL Data Types

COBOL Type	Java Type
66 RENAMES	not supported
66 RENAMES THRU	not supported
77 level	supported
88 level (<i>condition</i>)	not supported (ignored)
<i>group record</i>	inner class
OCCURS (<i>fixed array</i>)	array
OCCURS DEPENDING (<i>variable-length array</i>)	array
OCCURS INDEXED BY	not supported (ignored)
OCCURS KEY IS	not supported (ignored)

7 Programming Scenarios

This section contains three scenarios for developing and deploying BEA eLink Java Adapter for Mainframe WLS Edition (JAM) applications. The scenarios are based on the general procedures presented in “Developing Java Applications.” They give you practical examples for using JAM tools, presented as tasks with step-by-step procedures. They illustrate typical situations, but do not represent all possible uses of the product.

The scenarios depict the development of a new application and the updating of existing applications. WebLogic Server (WLS) samples are used to illustrate any existing applications. All discussions are from the application developer’s point of view, presume a properly installed and configured environment, and presume an appropriate mainframe application is available.

The following scenarios are provided:

- Scenario A: Developing a Multi-Service Data Entry Servlet
- Scenario B: Enhancing an Existing Servlet to Originate a Mainframe Request
- Scenario C: Updating an Existing EJB to Service a Mainframe Request

Note: Although the sample code in this section represents typical applications, it is intended for example only and is not supported for actual use.

Scenario A: Developing a Multi-Service Data Entry Servlet

In this scenario, you develop a multi-service application, as opposed to the single-service application presented in “Generating a Servlet-Only JAM Application.” The scenario contains more sophisticated data validation with tips on how to manage states and it includes cosmetic enhancements using static HTML frames to present default forms.

Task 1: Use EgenCobol to Create a Base Application

Prerequisite

You must first identify the mainframe application and obtain its COBOL copybook. This is typically a CICS DFHCOMAREA or the user data portion of an IMS queue record layout. The copybook’s name in this discussion is `emprec.cbl`, as shown in Listing 7-1.

Listing 7-1 Mainframe Application COBOL Copybook `emprec.cbl`

```
02 emp-record.
   05 emp-ssn                                pic 9(9) comp-3.
   05 emp-name.
       10 emp-name-last                      pic x(15).
       10 emp-name-first                     pic x(15).
       10 emp-name-mi                        pic x.
   05 emp-addr.
       10 emp-addr-street                    pic x(30).
       10 emp-addr-st                        pic x(2).
       10 emp-addr-zip                       pic x(9).
```

Step 1: Prepare EgenCobol Script

In Listing 7-2, the data view `emprecData` is generated from the copybook named `emprec.cbl`.

Listing 7-2 Basic EgenCobol script

```
view emprecData from emprec.cbl
```

Step 2: Add Service Entries

Add the single line service entries in Listing 7-3 for create, read, update, and delete operations. They all use `empRecData` as input and return `emprecData` as output. In this example, a single data view is used; however, the input and output data views could be different.

Listing 7-3 Service Names Associated with Input and Output Views

```
service empRecCreate accepts empRecData returns empRecData  
service empRecRead accepts empRecData returns empRecData  
service empRecUpdate accepts empRecData returns empRecData  
service empRecDelete accepts empRecData returns empRecData
```

Step 3: Add Page Declaration in EgenCobol Script

Multiple pages can be chained together. Any service entries should match services defined elsewhere in the script. The page declarations shown in Listing 7-4 associate buttons on the HTML display with services declared in the previous step.

Listing 7-4 Page Declaration Associating Display Buttons with Services

```
page empRecPage "Employee Record" {  
  view empRecData  
    buttons {  
      "Create" service(empRecCreate) shows empRecPage  
      "Read" service(empRecRead) shows empRecPage  
      "Update" service(empRecUpdate) shows empRecPage  
      "Delete" service(empRecDelete) shows empRecPage  
    }  
}
```

Step 4: Add Servlet Name

As shown in Listing 7-5, `empRecServlet` is the servlet name to be registered as a URL in the WLS properties file. (Every servlet requires a URL to be registered this way. Refer to WLS documentation about deploying servlets for more specific information.) Here, the `empRecPage` is to be displayed when the `empRecServlet` is invoked.

Listing 7-5 Add Servlet Name

```
servlet empRecServlet shows empRecPage
```

The script is then saved as `emprec.egen`.

Step 5: Generate the Java Source Code

In Listing 7-6, invoke the `EgenCobol` code generator to create the base application that is then compiled. This makes class files (`*.class`) available for servlet customizing. The `empRecData.java` is the data view object for `emprec.cbl`.

Warning: `CLASSPATH` should include the WLS subdirectories and the `jam.jar` file; otherwise, the compile fails.

Note: You can create a script file containing the `EgenCobol` command line, along with the `javac` command to make the invocation easier.

Listing 7-6 Generating the Java Source Code

```
java bea.jam.egen.EgenCobol empreg.egen
ls emp*.java
    empRecData.java      empRecServlet.java

javac emp*.java
```

Step 6: Review the Java Source Code

It is a good idea to obtain a list of accessors for use later. In general, you should look at the `EgenCobol` output to become familiar with each of the scenarios presented in this section.

The entire method of customizing the generated output is predicated on derivation from generated code. The base application can be regenerated without destroying the custom code.

Note: Each COBOL group item has its own accessor. This is important because the group name represents a nested inner class that must be accessed in order to retrieve the members.

In the Listing 7-7, the output from the `grep` command shows the relationships in reverse order, for example:

```
getEmpRecord().getEmpAddr().getEmpAddrSt()
```

This is illustrated in the actual code example shown subsequently in this scenario.

Listing 7-7 Review the Java Source Code

```
grep get emp*.java
empRecData.java:    public BigDecimal    getEmpSsn()
empRecData.java:    public String        getEmpNameLast()
empRecData.java:    public String        getEmpNameFirst()
empRecData.java:    public String        getEmpNameMi()
empRecData.java:    public EmpNameV    getEmpName()
empRecData.java:    public String        getEmpAddrStreet()
empRecData.java:    public String        getEmpAddrSt()
empRecData.java:    public String        getEmpAddrZip()
empRecData.java:    public EmpAddrV    getEmpAddr()
empRecData.java:    public EmpRecordV    getEmpRecord()
```

Task 2: Create Your Custom Application from the Base Application

The preferred customizing method is to derive a custom class from the generated base application.

Step 1: Start with Imports

In Listing 7-8, `BigDecimal` supports COMP-3 packed data. `HttpSession` is available for saving limited state. `DataView` is the base for `empRecData`. The `empRecData` and `empRecServlet` were generated from the COBOL copybook.

Listing 7-8 Using Imports to Start Creating the Custom Application

```
import java.math.BigDecimal;
import javax.servlet.http.HttpSession;
import bea.dmd.dataview.DataView;
import empRecData;
import empRecServlet;
```

Step 2: Declare the New Custom Class

Listing 7-9 shows how to extend the generated servlet. This enables regeneration of the base application without destroying customized code. Fields can be added to the copybook without disrupting the customized code.

Listing 7-9 Declaring the New Custom Class

```
public class customCrud
    extends empRecServlet
{
:
:
}
```

Step 3: Add Implementation for doGetSetup

In Listing 7-10, you can see how to provide a new data view and the http session. The HttpSession (s) can be used to hold a reference to the data view. This ensures you are actually in the first pass rather than a browser back arrow. The data view provided (dv) is a fresh instance of the empRecData data view.

Listing 7-10 Add Implementation for doGetSetup

```
public DataView doGetSetup(DataView dv, HttpSession s){
empRecData erd = (empRecData)s.getValue("customCrud");
if (erd == null)
    erd = (empRecData)dv; // use new dataview
}
```

Step 4: Continue Implementation for doGetSetup

In Listing 7-11, note the use of group level accessors to obtain fields. This code pre-fills fields with data entry hints as to what fields are required, or how numeric values should be entered. You can fill form data in any manner required prior to displaying.

Listing 7-11 Continue Implementation for doGetSetup

```
if(erd.getEmpRecord().getEmpSsn().compareTo(BigDecimal.valueOf(0L)) == 0)
    erd.getEmpRecord().setEmpSsn(BigDecimal.valueOf(123121234L));
if (erd.getEmpRecord().getEmpName().getEmpNameLast().length() == 0)
    erd.getEmpRecord().getEmpName().setEmpNameLast("Entry Required");
if (erd.getEmpRecord().getEmpName().getEmpNameFirst().trim().length() == 0)
    erd.getEmpRecord().getEmpName().setEmpNameFirst("Entry Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrStreet().trim().length() == 0)
    erd.getEmpRecord().getEmpAddr().setEmpAddrStreet("Entry Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrSt().trim().length() == 0)
    erd.getEmpRecord().getEmpAddr().setEmpAddrSt("TX");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrZip().trim().length() == 0)
    erd.getEmpRecord().getEmpAddr().setEmpAddrZip("123451234");
```

Step 5: Finish Implementation for doGetSetup

In Listing 7-12, note the use of the http session putValue to save a reference to the data view. The doGet() processing continues on return. This data will be presented in the displayed form.

Listing 7-12 Finish Implementation for doGetSetup

```
s.putValue("customCrud", (Object)erd);
    return erd;
}
```

Step 6: Create Implementation for doPostSetup

In Listing 7-13, the data view passed in contains values entered into the form by the application user. (The HttpSession is also available for use at this point, if required.)

Listing 7-13 Create Implementation for doPostSetup

```
public DataView doPostSetup(DataView dv, HttpSession s)
{
    empRecData erd = (empRecData)dv;
```

Step 7: Continue Implementation for doPostSetup

In Listing 7-14, note the use of group-level accessors to obtain fields. This code checks for original defaults, as well as missing data. The `SocialSecurity` is a `BigDecimal` object. Validation can be simple or complex as necessary.

Listing 7-14 Continue implementation for doPostSetup

```
if(erd.getEmpRecord().getEmpSsn().compareTo(BigDecimal.valueOf(0L)) == 0)
    throw new Error("Social Security Number Is Required");
if(erd.getEmpRecord().getEmpSsn().compareTo(BigDecimal.valueOf(123121234L)) == 0)
    throw new Error("Social Security Number Is Required");
if (erd.getEmpRecord().getEmpName().getEmpNameLast() == null)
    throw new Error("Last Name Is Required");
if (erd.getEmpRecord().getEmpName().getEmpNameLast().trim().length() == 0)
    throw new Error("Last Name Is Required");
if (erd.getEmpRecord().getEmpName().getEmpNameLast().trim().compareTo("Entry
    Required") == 0)
    throw new Error("Last Name Is Required");
```

Step 8: Continue Implementation of doPostSetup

In Listing 7-15, note the use of group-level accessors to obtain fields. This code checks for original defaults, as well as missing data. (Validation routines could have been split out by field.)

Listing 7-15 Continue Implementation of doPostSetup

```
if (erd.getEmpRecord().getEmpName().getEmpNameFirst() == null)
    throw new Error("First Name Is Required");
if (erd.getEmpRecord().getEmpName().getEmpNameFirst().trim().length() == 0)
    throw new Error("First Name Is Required");
if (erd.getEmpRecord().getEmpName().getEmpNameFirst()
    .trim().compareTo("Entry Required") == 0)
    throw new Error("First Name Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrStreet() == null)
    throw new Error("Street Address Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrStreet().trim().length() == 0)
    throw new Error("Street Address Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrStreet()
    .trim().compareTo("Entry Required") == 0)
    throw new Error("Street Address Is Required");
```

Step 9: Continue Implementation for doPostSetup

In Listing 7-16, notice the use of group-level accessors to obtain fields. This code checks for original defaults, as well as missing data. Depending on the application, it may be more advantageous to develop validations as separate methods. This enables routines to be developed and tested with a servlet and easily re-used in an EJB.

Listing 7-16 Continue Implementation for doPostSetup

```
if (erd.getEmpRecord().getEmpAddr().getEmpAddrSt() == null)
    throw new Error("State Abbreviation Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrSt().trim().length() == 0)
    throw new Error("State Abbreviation Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrSt()
    .trim().compareTo("TX") != 0)
    throw new Error("Texas Employees ONLY");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrZip() == null)
    throw new Error("ZipCode Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrZip().trim().length() == 0)
    throw new Error("ZipCode Is Required");
if (erd.getEmpRecord().getEmpAddr().getEmpAddrZip()
    .trim().compareTo("123451234") == 0)
    throw new Error("ZipCode Is Required");
```

Step 10: Finish Implementation of doPostSetup

In Listing 7-17, the http session `getValue` is used to remove a reference to the data view. This prevents re-posting the same data twice. The `doPost` processing continues on return. This data is now passed to the mainframe.

Listing 7-17 Finish Implementation for doPostSetup

```
else
    s.removeValue("customCrud");
    return erd;
}
```

Step 11: Create Implementation for doPostFinal

In Listing 7-18, the `doPostFinal` occurs after mainframe transmission, but prior to re-display in the browser. This example clears entered data after it is sent to the mainframe. This step completes the custom servlet.

Listing 7-18 Create Implementation for doPostFinal

```
public DataView doPostFinal(DataView dv, HttpSession s){
    empRecData erd = (empRecData)dv;
    erd.getEmpRecord().setEmpSsn(BigDecimal.valueOf(0L));
    erd.getEmpRecord().getEmpName().setEmpNameLast("");
    erd.getEmpRecord().getEmpName().setEmpNameFirst("");
    erd.getEmpRecord().getEmpName().setEmpNameMi("");
    erd.getEmpRecord().getEmpAddr().setEmpAddrStreet("");
    erd.getEmpRecord().getEmpAddr().setEmpAddrSt("");
    erd.getEmpRecord().getEmpAddr().setEmpAddrZip("");
    return erd; }

```

Step 12: Update the jcrmgw.cfg File with Service Entries

Listing 7-19 defines the entries which are used when the corresponding Create/Read/Update/Delete form buttons are pushed; for example, the Create button triggers `empRecCreate` which invokes `DPLDEMOC`. The gateway must be restarted for the new services to take effect.

Listing 7-19 Update jcrmgw.cfg File

<code>empRecCreate</code>	<code>RDOM="CICS410"</code> <code>RNAME="DPLDEMOC"</code>
<code>empRecRead</code>	<code>RDOM="CICS410"</code> <code>RNAME="DPLDEMOR"</code>
<code>empRecUpdate</code>	<code>RDOM="CICS410"</code> <code>RNAME="DPLDEMOU"</code>
<code>empRecDelete</code>	<code>RDOM="CICS410"</code> <code>RNAME="DPLDEMOD"</code>

Step 13: Create Basic Three-Part HTML Frame

In Listing 7-20, the primary frame (identified as “main” in the HTML code) displays the servlet, while an auxiliary frame provides links to HELP pages. The “Built on BEA WebLogic” logo is also displayed. A single line of Java script is used to ensure the window displays in the foreground.

Listing 7-20 Create Basic Three-Part HTML Frame

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>eGen</title>
  </head>
  <script language="javascript">
    <!--
    if (window.focus) {self.focus();} // -->
  </script>
  <FRAMESET cols="20%, 80%">
    <FRAMESET rows="20%, 80%">
      <FRAME src="bea_built_on_wl.gif" name="logo">
      <FRAME src="panel.html" name="aux">
    </FRAMESET>
    <FRAME src="http://machine.domain.com:7001/empRec" name="main">
  </FRAMESET>
</html>
```

Step 14: Create a Series of Links to HELP Pages

Listing 7-21 shows how the HTML can display as a sidebar frame. The `intro.html`, `emprec.html`, and `create.html` can display inside the “main” frame to provide basic HELP.

Listing 7-21 Creating a Series of HELP Page Links

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head> <title>eGen help</title> </head>
  <script language="javascript">
    <!--
    if (window.focus) {self.focus();} // -->
  </script>
  <body>
    <TABLE summary="This table contains links to help pages.">
      <TR> <TH>empRec Info</TH>
      <TR> <TD><a href="intro.html" target="help">Introduction </a>
      <TR> <TD><a href="emprec.html" target="help">EmpRec </a>
      <TR> <TD><a href="create.html" target="help">Create </a>
      <TR> <TD><a href="read.html" target="help">Read </a>
      <TR> <TD><a href="update.html" target="help">Update </a>
      <TR> <TD><a href="delete.html" target="help">Delete </a>
    </TABLE>
  </body>
</html>
```

Task 3: Update the JAM Configurations and Update WLS Properties

Update the `jcrmgw.cfg` file with the remote service entries shown in Listing 7-22. The Java gateway must be restarted for new services. The entries are used when the corresponding form button is pushed. The `Create` button triggers `empRecCreate`, which invokes `DPLDEMO`. The service name must match values in the `EgenCobol` script. In this example, the `RNAME` must match an actual CICS program name.

Listing 7-22 Remote Service Entries for Create/Read/Update/Delete

empRecCreate	RDOM="CICS410 " RNAME="DPLDEMOC"
empRecRead	RDOM="CICS410 " RNAME="DPLDEMOR"
empRecUpdate	RDOM="CICS410 " RNAME="DPLDEMOU"
empRecDelete	RDOM="CICS410 " RNAME="DPLDEMOM"

Update the `weblogic.properties` file with the entries shown in the Listing 7-23.

Listing 7-23 Update WLS Properties File

```
weblogic.httpd.register.customEmpRec=customCrud
```

Task 4: Deploy Your Application

At this point, you have a basic form capable of receiving data entry, along with some static HTML code for display. The following are standard WLS servlet deployment steps:

1. If required, compile the *.java files.
2. Copy the class files to the WLS servlet directory.
3. If required, add URL registration lines to the `weblogic.properties` file.
4. If required, restart WLS.
5. Test the result with your browser pointed at the registered URL.

Task 5: Use the Application

Figure 7-1 shows the default servlet with customized code displayed in an HTML facade. This type of servlet is useful for presentation, proof-of-concept, and as a test bed for development.

Figure 7-1 New Data Entry Servlet Display

The screenshot shows a Netscape browser window with the address bar displaying `http://dalton2.bea.com:7801/egen/emprec/egen.html`. The page title is "empRecord". On the left, a sidebar contains a "Built On" logo for BEA WebLogic and a list of links: "empRec Info", "Introduction", "EmpRec", "Create", "Read", "Update", and "Delete". The main content area is a form with three sections: "empSsn" with a text field containing "123121234"; "empName" with three text fields: "empNameLast" (Entry Required), "empNameFirst" (Entry Required), and "empNameMi" (empty); and "empAddr" with three text fields: "empAddrStreet" (Entry Required), "empAddrSt" (TX), and "empAddrZip" (123451234). At the bottom of the form are buttons for "Create", "Read", "Update", "Delete", "Clear Form", and "Refresh Form".

Figure 7-2 shows the servlet with the Create HELP page displayed in a new window over the application.

Figure 7-2 Servlet with HELP Page Displayed

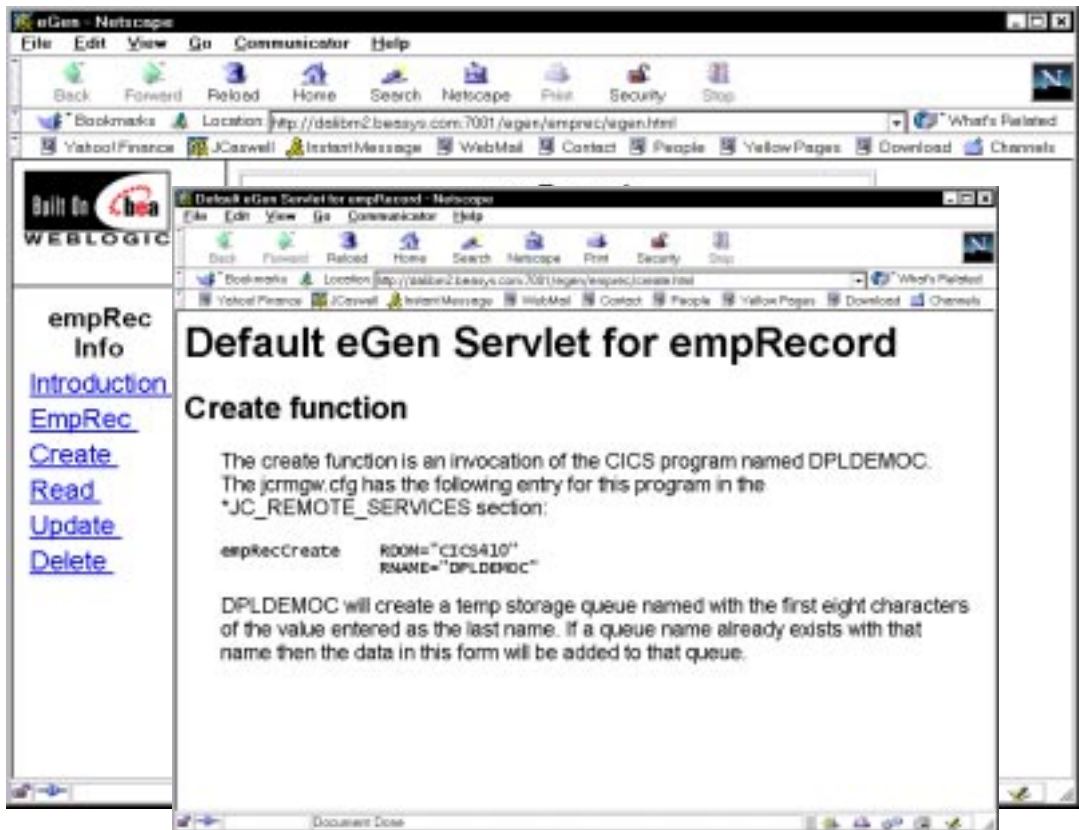


Figure 7-3 is an example of the page used for the front end of the new custom servlet.

Figure 7-3 New Data Entry Servlet Front End Page



Scenario A Summary

The following listings show COBOL programs for each of the button and service combinations:

- Create (DPLDEMOC)
- Read (DPLDEMOR)
- Update (DPLDEMOU)
- Delete (DPLDEMOD)

All of these programs make use of a CICS temporary storage queue for data. This is a simple technique that is useful for testing and demonstrations.

Create

The simple program shown in Listing 7-24 writes a temporary storage queue using the first eight characters of the employee name as the QID.

Listing 7-24 COBOL Program for Create (DPLDEMOC)

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.      DPLDEMOC.
    INSTALLATION.
    DATE-COMPILED.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  TSQ-DATA-LENGTH      PIC S9(4) COMP VALUE ZERO.
    01  TSQ-NAME.
        05 TSQ-ID            PIC X(8) VALUE SPACES.
        05 FILLER            PIC X(30) VALUE SPACES.
    LINKAGE SECTION.
    01  DFHCOMMAREA.
        COPY EMPREC.
    PROCEDURE DIVISION.
    MAINLINE SECTION.
        MOVE EMP-NAME TO TSQ-NAME
        MOVE LENGTH OF EMP-RECORD
        TO TSQ-DATA-LENGTH
        EXEC CICS WRITEQ TS
            QUEUE(TSQ-ID)
            FROM(EMP-RECORD)
            LENGTH(TSQ-DATA-LENGTH)
        END-EXEC.
        EXEC CICS RETURN
        END-EXEC.
        EXIT.
```

Read

The simple program shown in Listing 7-25 reads a temporary storage queue using the first eight characters of the employee name as the QID. If the read fails, the COMMAREA is reset in consideration of the client application so residual data does not appear as the result of a read.

Listing 7-25 COBOL Program for Read (DPLDEMOR)

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.    DPLDEMOR.
    INSTALLATION.
    DATE-COMPILED.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  TSQ-DATA-LENGTH          PIC S9(4) COMP VALUE ZERO.
    01  TSQ-RESP                 PIC S9(4) COMP VALUE ZERO.
    01  TSQ-NAME.
        05  TSQ-ID              PIC X(8) VALUE SPACES.
        05  FILLER              PIC X(30) VALUE SPACES.
    LINKAGE SECTION.
    01  DFHCOMMAREA.
        COPY EMPREC.
    PROCEDURE DIVISION.
    MAINLINE SECTION.
        MOVE EMP-NAME TO TSQ-NAME
        MOVE LENGTH OF EMP-RECORD
        TO TSQ-DATA-LENGTH
        EXEC CICS READQ  TS
                ITEM(1)
                INTO(EMP-RECORD)
                QUEUE(TSQ-ID)
                LENGTH(TSQ-DATA-LENGTH)
                RESP(TSQ-RESP)
        END-EXEC.
        IF TSQ-RESP NOT EQUAL ZERO
            MOVE ZEROS  TO EMP-SSN
            MOVE SPACES TO EMP-NAME-FIRST
            MOVE SPACES TO EMP-NAME-MI
            MOVE SPACES TO EMP-ADDR
        END-IF
        EXEC CICS RETURN
    END-EXEC.
```

Update

The simple program shown in Listing 7-26 deletes a temporary storage queue using the first eight characters of the employee name as the QID. It then creates a new queue with the COMMAREA provided.

Listing 7-26 COBOL Program for Update (DPLDEMOU)

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.    DPLDEMOU.
    INSTALLATION.
    DATE-COMPILED.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  TSQ-DATA-LENGTH          PIC S9(4) COMP VALUE ZERO.
    01  TSQ-NAME.
        05  TSQ-ID              PIC X(8) VALUE SPACES.
        05  FILLER              PIC X(30) VALUE SPACES.
    LINKAGE SECTION.
    01  DFHCOMMAREA.
        COPY EMPREC.
    PROCEDURE DIVISION.
    MAINLINE SECTION.
        MOVE EMP-NAME TO TSQ-NAME
        MOVE LENGTH OF EMP-RECORD
        TO TSQ-DATA-LENGTH
        EXEC CICS DELETEQ TS
            QUEUE(TSQ-ID)
        END-EXEC.
        EXEC CICS WRITEQ TS
            QUEUE(TSQ-ID)
            FROM(EMP-RECORD)
            LENGTH(TSQ-DATA-LENGTH)
        END-EXEC.
        EXEC CICS RETURN
        END-EXEC.
        EXIT.
```

Delete

This simple program shown in Listing 7-27 deletes a temporary storage queue using the first eight characters of the employee name as the QID. The COMMAREA is reset in consideration of the client application so residual data does not remain after the delete.

Listing 7-27 COBOL Program for Delete (DPLDEMOD)

```
IDENTIFICATION DIVISION.  
    PROGRAM-ID.      DPLDEMOD.  
    INSTALLATION.  
    DATE-COMPILED.  
    ENVIRONMENT DIVISION.  
    DATA DIVISION.  
    WORKING-STORAGE SECTION.  
    01  TSQ-DATA-LENGTH          PIC S9(4) COMP VALUE ZERO.  
    01  TSQ-NAME.  
        05  TSQ-ID              PIC X(8) VALUE SPACES.  
        05  FILLER              PIC X(30) VALUE SPACES.  
    LINKAGE SECTION.  
    01  DFHCOMMAREA.  
        COPY EMPREC.  
    PROCEDURE DIVISION.  
    MAINLINE SECTION.  
        MOVE EMP-NAME TO TSQ-NAME  
        MOVE LENGTH OF EMP-RECORD  
        TO  TSQ-DATA-LENGTH  
        EXEC CICS DELETEQ TS  
            QUEUE(TSQ-ID)  
        END-EXEC.  
        MOVE SPACES  
        TO  DFHCOMMAREA  
        MOVE ZEROS  TO EMP-SSN  
        EXEC CICS RETURN  
        END-EXEC.  
    EXIT.
```

Scenario B: Enhancing an Existing Servlet to Originate a Mainframe Request

In this scenario, you take the WebLogic Server (WLS) `survey` servlet and add a mainframe request to the `post` routine. You add the code to the `postprocessing` routine, creating a mainframe buffer and sending it to CICS where an application writes the buffer to a temporary storage queue and returns.

Task 1: Use EgenCobol to Create a Base Class

Prerequisite

You should have successfully created the `survey` servlet prior to attempting the enhancement discussed in this scenario. You must then identify the mainframe application and obtain its COBOL copybook. This is typically a CICS `DFHCOMAREA` or the user data portion of an IMS queue record layout. The copybook's name in this discussion is `survey.cbl`, shown in Listing 7-28.

Listing 7-28 Mainframe Application COBOL Copybook `survey.cbl`

```
02  survey-record.
    05  survey-ide          pic x(12).
    05  survey-emp          pic x(12).
    05  survey-cmt          pic x(256).
```

Step 1: Prepare EgenCobol Script

In Listing 7-29, both the data view `surveyData` and the client class `SurveyClient` are generated from the copybook `survey.cbl`.

Listing 7-29 Basic EgenCobol script

```
view surveyData from survey.cbl
service doSurvey accepts surveyData returns surveyData
client class SurveyClient
{
    method doSurvey is service doSurvey
}
```

You are now finished creating the `survey.egen` script file and are ready to generate the source code.

Step 2: Generate the Java Source Code

In Listing 7-30, you invoke the EgenCobol code generator to create the base class that is then compiled. This makes class files (*.class) available for servlet customizing. The `surveyData.java` is the data view object for `survey.cbl`.

Warning: CLASSPATH should have both the WLS subdirectories and the `jam.jar` file; otherwise, the compile fails.

Note: You could create a script file containing the EgenCobol command line, along with the `javac` command to make the invocation easier.

Listing 7-30 Generating the Java Source Code

```
java bea.jam.egen.EgenCobol survey.egen
ls *.java
SurveyServlet.java surveyData.java SurveyClient.java
javac *.java
```

Step 3: Review the Java Source Code

It is a good idea to obtain a list of accessors for use later. In general, you should look at the EgenCobol output to become familiar with each of the scenarios presented in this section.

Note: Each COBOL group item has its own accessor. This is important because the group name represents a nested inner class that must be accessed in order to retrieve the members.

In Listing 7-31, the output from the `grep` command shows the relationships in reverse order, for example:

```
getSurveyRecord().getSurveyIde()
```

This is illustrated in the actual code example shown subsequently in this scenario.

Listing 7-31 Review the Java Source Code

```
grep get surveyData.java
    public String      getSurveyIde()
    public String      getSurveyEmp()
    public String      getSurveyCmt()
    public SurveyRecordV getSurveyRecord()
grep set surveyData.java
    public void setSurveyIde(String value)
    public void setSurveyEmp(String value)
    public void setSurveyCmt(String value)
```

Task 2: Update the Survey Servlet Using the Generated Class

The preferred customizing method is to derive a custom class from the generated base application. You are now ready to update the WLS example `survey` servlet.

Step 1: Start with Imports

In Listing 7-32, `bea.jam.egen` provides the `EgenCobol` client and data view base. The `surveyData` is the specific data view generated from the COBOL copybook. `SurveyClient` is the generated client class.

Listing 7-32 Using Imports to Start Creating the Custom Application

```
import bea.jam.egen.*;
import surveyData;
import SurveyClient;
```

Step 2: Add New Data Members

In Listing 7-33, the code adds a private member for `SurveyClient`, which can be created in the `init()` function because there is no state for it. The `init()` is then updated for a new member. The `SurveyClient` obtains a connection factory when created. A single instance of `SurveyClient` can serve all requests.

Listing 7-33 Adding New Data Members

```
//Add private member for SurveyClient
private SurveyClient egc = null;
//Update init() for new member
egc = new SurveyClient();
```

Step 3: Update doPost with Mainframe Request

In Listing 7-34, add the local variables for form data and data view in `doPost`. The data view is the minimum requirement. The `values` entry has been declared previously.

Listing 7-34 Update doPost with Mainframe Request

```
values = req.getParameterNames();
surveyData sd = new surveyData();
```

Step 4: Continue Updating doPost by Extracting Form Data

In Listing 7-35, the code loops through the form using data view accessors to set data. The submit field is skipped. The surveyData accessors are used to set values for ide, employee, and comment. The surveyData object represents the mainframe message buffer that ultimately is used to make the request. (The surveyData class was generated using the EgenCobol code generator with the mainframe COBOL copybook.)

Listing 7-35 Continue Updating doPost

```
while(values.hasMoreElements()) {
    String name = (String)values.nextElement();
    String value = req.getParameterValues(name)[0];
    if(name.compareTo("submit") != 0) {
        if(name.compareTo("ide") == 0)
            sd.getSurveyRecord().setSurveyIde(value);
        else if(name.compareTo("employee") == 0)
            sd.getSurveyRecord().setSurveyEmp(value);
        else if(name.compareTo("comment") == 0)
            sd.getSurveyRecord().setSurveyCmt(value);
    }
}
```

Step 5: Continue Updating doPost by Calling Mainframe Service

In Listing 7-36, the code shows how to make the mainframe request. The doSurvey command blocks until a response is provided. The call can throw either `IOException` or `snaException`. In this listing, doSurvey is in a try block that catches `IOException`. The doSurvey command returns a data view that contains any response.

Listing 7-36 Continue Updating doPost

```
egc.doSurvey(sd);
```

The `snaException` is the base class for several exceptions, shown in Listing 7-37. A time-out is the most likely error an application would get.

Listing 7-37 Mainframe Exceptions

```
snaException
    jcrmConfigurationException
    snaCallFailureException
    snaLinkNotFoundException
    snaNoSessionAvailableException
    snaRequestTimeoutException
    snaServiceNotReadyException
```

Task 3: Update the JAM Configurations and Update WLS Properties

In Listing 7-38, update the `jcrmgw.cfg` file with the remote service name `doSurvey`. The Java gateway must be restarted for new services to take effect. The `RNAME DPLSURVY` is a CICS program that exists on the mainframe.

Listing 7-38 Update the `jcrmgw.cfg` File with Service Name

```
doSurvey      RDOM= "CICS410 "
              RNAME= "DPLSURVY "
```

Update the `weblogic.properties` file with the entries shown in Listing 7-39.

Listing 7-39 Update WLS Properties File

```
weblogic.httpd.register.survey=examples.servlets.SurveyServlet
```

Task 4: Deploy Your Application

At this point, you have a basic form capable of making a maintenance request. The following are standard WLS servlet deployment steps:

1. If required, compile *.java files.
2. Copy class files to the WLS servlet directory.
3. Copy HTML form to WLS document directory.
4. If required, add URL registration to `weblogic.properties` file.
5. If required, restart WLS.
6. Test the result with your browser pointed at the registered URL.

Scenario B Summary

The simple program shown in Listing 7-40 writes the contents of the COMMAREA to a temporary storage queue. This type of servlet is useful for testing, demonstrations, and new application development.

Listing 7-40 COBOL Program for DPLSURVY

```
IDENTIFICATION DIVISION.  
    PROGRAM-ID.      DPLSURVY.  
    INSTALLATION.  
    DATE-COMPILED.  
    ENVIRONMENT DIVISION.  
    DATA DIVISION.  
    WORKING-STORAGE SECTION.  
    01  TSQ-DATA-LENGTH      PIC S9(4) COMP VALUE ZERO.  
    01  TSQ-ID               PIC X(8) VALUE SPACES.  
    LINKAGE SECTION.  
    01  DFHCOMMAREA.  
        COPY SURVEY.  
    PROCEDURE DIVISION.  
    MAINLINE SECTION.  
        MOVE 'SURVEY' TO TSQ-NAME  
        MOVE LENGTH OF SURVEY-RECORD  
        TO  TSQ-DATA-LENGTH  
        EXEC CICS WRITEQ TS  
            QUEUE(TSQ-ID)  
            FROM(SURVEY-RECORD)  
            LENGTH(TSQ-DATA-LENGTH)  
        END-EXEC.  
        EXEC CICS RETURN  
        END-EXEC.  
        EXIT.
```

Note: Some applications have extremely large COMMAREA copybooks. Distributed applications can be very sensitive to large amounts of data being transferred between components. If the Java application requires only a few fields from a large copybook, it would be advantageous to front-end the target application with a simpler program passing only the data required.

Scenario C: Updating an Existing EJB to Service a Mainframe Request

In this scenario, you take the WebLogic Server (WLS) basic `statelessSessionTraderBean` and update the interface to add a dispatch function that is given control upon receipt of an inbound request. The `EgenCobol` client class code generation model is used. The `TraderBean` is designed to run from a stand-alone client and output a list of stock trades.

Task 1: Use EgenCobol to Create a Base Class

Prerequisite

You should have successfully run the WLS basic `statelessSessionTraderBean` prior to attempting the updates discussed in this scenario. You must then identify the mainframe application and obtain its COBOL copybook. This is typically a CICS `DFHCOMAREA` or the user data portion of an IMS queue record layout. The copybook's name in this discussion is `trader.cbl`, as shown in Listing 7-41.

Listing 7-41 Mainframe Application COBOL Copybook `trader.cbl`

```
02  TRADER-RECORD.
    05  CUSTOMER                PIC X(24).
    05  SYMBOL                  PIC X(6).
    05  SHARES                  PIC 9(7) COMP-3.
    05  PRICE                   PIC 9(7) COMP-3.
```

Step 1: Prepare EgenCobol Script

The single-line script in Listing 7-42 generates the data view `traderData` from the copybook named `trader.cbl`. The script is then saved as `inboundEJB.egen`.

Listing 7-42 Basic EgenCobol script

```
view traderData from trader.cbl
```

You are now finished creating the `inboundEJB.egen` script file and are ready to generate the source code.

Step 2: Generate the Java Source Code

In Listing 7-43, you invoke the EgenCobol code generator to compile `trader.cbl` copybook and `inboundEJB.egen`. The `traderData.java` is the data view object for `trader.cbl`.

Warning: `CLASSPATH` should have both the WLS subdirectories and the `jam.jar` file; otherwise, the compile fails.

Note: You could create a script file containing the EgenCobol command line, along with the `javac` command to make the invocation easier.

Listing 7-43 Generating the Java Source Code

```
java bea.jam.egen.EgenCobol inboundEJB.egen
ls traderDat*.java
traderData.java
javac traderData.java
```

Step 3: Review the Java Source Code

It is a good idea to obtain a list of accessors for use later. Look at the EgenCobol output to become familiar with each of the scenarios presented in this section.

The entire method of customizing the generated output is predicated on deriving the output from generated code. The base application can be regenerated without destroying the custom code.

Note: Each COBOL group item has its own accessor. This is important because the group name represents a nested inner class that must be accessed in order to retrieve the members.

In Listing 7-44, the output from the `grep` command shows the relationships in reverse order, for example:

```
getTraderRecord().getPrice()
```

This is illustrated in the actual code example shown subsequently in this scenario.

Listing 7-44 Review the Java Source Code

```
grep get traderData.java
    public String      getCustomer()
    public String      getSymbol()
    public BigDecimal  getShares()
    public BigDecimal  getPrice()
    public TraderRecordV getTraderRecord()
grep set traderData.java
    public void setCustomer(String value)
    public void setSymbol(String value)
    public void setShares(BigDecimal value)
    public void setPrice(BigDecimal value)
```

Task 2: Update the Trader Interface Using the Generated Class

You are now ready to update the WLS `trader` example basic `statelessSession` bean.

Step 1: Start with Import

In Listing 7-45, the EJB interface is updated. In the `Trader` interface declaration, the `EJBObject` is replaced with `gwObject`. The `gwObject` extends `EJBObject` and provides the `dispatch` method that gets control on receipt of an incoming request.

Listing 7-45 Using Imports to Start Updating the EJB

```
import bea.sna.jcrmgw.gwObject;  
.  
.  
.  
public interface Trader extends gwObject {  
.  
.  
.
```

Step 2: Continue with Imports

In Listing 7-46, you perform four imports to update the EJB. The `bea.base.io.*` import provides the mainframe reader and writer. The `traderData` import is the specific data view generated from the COBOL copybook. The `BigDecimal` class handles packed decimal COMP-3 fields. The mainframe reader and writer can generate `IOExceptions`.

Listing 7-46 Continuing Imports

```
import bea.base.io.*;  
import traderData;  
import java.math.BigDecimal;  
import java.io.IOException;
```

Step 3: Update EJB with dispatch

In Listing 7-47, the gateway invokes `dispatch` with a byte array of mainframe EBCDIC data. The code converts the mainframe byte array to a data view using a `MainFrameReader`. The `traderData` is the generated data view class.

Listing 7-47 Update EJB with dispatch

```
.
.
.
public byte[] dispatch(byte[] b)
{
    traderData td = null;
    try {
        td = new traderData(new MainframeReader(b));
    } catch(IOException ie) { return b; }
        // error protocol required
}
```

Step 4: Continue Updating EJB with dispatch

In Listing 7-48, the code uses accessors to get input and set output. The mainframe COMMAREA is updated with the result. Note the use of an accessor to obtain the group level class prior to accessing the member variable. An application level error indicator in the data is used to convey the exception. Updating the data view member results in updates to the mainframe application. Any application exception thrown from the dispatch routine results in an abend returned to the mainframe.

Listing 7-48 Continue Updating EJB with dispatch

```
try {
    TradeResult tr = buy(td.getTraderRecord().getCustomer()
        ,td.getTraderRecord().getSymbol()
        ,td.getTraderRecord().getShares().intValue());
    td.getTraderRecord().setShares(new
        BigDecimal((long)tr.numberTraded));
    td.getTraderRecord().setPrice(new
        BigDecimal((long)tr.priceSoldAt));
} catch(ProcessingErrorException pe)
    td.getTraderRecord().setSymbol(" *ERROR"); }
```

Step 5: Finish Updating EJB with dispatch

In Listing 7-49, the code converts the data view back into a byte array to be returned to the mainframe using a `MainframeWriter`. The `MainframeWriter` and data view handle conversions. Note that the `dispatch` function takes a byte array and returns a byte array. This means when you set up an initial configuration, you can stub `dispatch` as an echo function.

Listing 7-49 Finish Updating EJB with dispatch

```
try {
    return td.toByteArray(new MainframeWriter());
} catch(IOException ie) {return b; }
                                // error protocol required
}
```

Task 3: Update the JAM Configurations

Update the `jcrmgw.cfg` file with the service name shown in Listing 7-50. The Java gateway must be restarted for new services to take effect.

Listing 7-50 Update the `jcrmgw.cfg` File with Service Name

```
*JC_LOCAL_SERVICES
statelessSession.TraderHome      RNAME="DPL1SVR"
```

Task 4: Deploy Your Application

Use the build function supplied with WLS to build the basic `statelessSession` example. The EJB is saved in `/myserver/ejb_basic_statelessSession.jar`. To deploy the EJB, either use the hot deploy feature of the WLS console, or add an entry to deploy the jar file in the `weblogic.ejb.deploy` property in the `weblogic.properties` file.

To run the client, follow the instructions in the WLS documentation.

Warning: Data view classes are not included in the jar file using the default script. You must either add `traderData*.class` entries to the jar file, or copy the entries to another location on the CLASSPATH. The EJB does not deploy if the `traderData` classes cannot be found.

Task 5: Use the Application

Listing 7-51 shows the inbound mainframe request for a “buy” transaction executed by the `traderBean`. If the previous tasks have been performed correctly, the result should look similar to this listing.

Listing 7-51 Inbound Mainframe Request

```
Thu Feb 17 15:31:10 CST 2000:<I> <EJB> EJB home interface:
'examples.ejb.basic.statelessSession.TraderHome' deployed bound to
the JNDI name: 'statelessSession.TraderHome'

Thu Feb 17 15:31:10 CST 2000:<I> <EJB> 0 EJBs were deployed using
.ser files.

Thu Feb 17 15:31:10 CST 2000:<I> <EJB> 1 EJBs were deployed using
.jar files.
.
.
.

**** Inbound Mainframe Request ****

buy (JEFF TESTER, WEBL, 150)

Executing stmt: insert into tradingorders (account, stockSymbol,
shares, price) VALUES ('JEFF TESTER','WEBL',150,10.0)
```

Scenario C Summary

The simple program shown in Listing 7-52 writes the contents of the COMMAREA to a temporary storage queue. This type of simple mainframe program is useful for testing, demonstrations, and new application development.

Listing 7-52 COBOL Program for DPL1CLT

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.      DPL1CLT.
    INSTALLATION.
    DATE-COMPILED.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01  STUFF.
        COPY INBOUND.
    PROCEDURE DIVISION.
    MAINLINE SECTION.
        MOVE 'JEFF TESTER' TO CUSTOMER
        MOVE 'WEBL'        TO SYMBOL
        MOVE ZEROS          TO PRICE
        MOVE +150           TO SHARES
        EXEC CICS LINK
            PROGRAM( 'DPL1SVR' )
            COMMAREA( STUFF )
        END-EXEC.
        EXEC CICS WRITEQ TS
            QUEUE( 'TRADER' )
            FROM( STUFF )
        END-EXEC.
        EXEC CICS RETURN
        END-EXEC.
```

Note: Some applications have extremely large COMMAREA copybooks. Distributed applications can be very sensitive to large amounts of data being transferred between components. If the Java application requires only a few fields from a large copybook, it would be advantageous to preface the target application with a simpler program passing only the data required.

A Code Generator Reference Pages

This appendix contains reference pages for the eLink Java Adapter for Mainframe WLS Edition COBOL code generator (EgenCobol). It describes the rules for writing the script file that controls the code generator.

EgenCobol

The EgenCobol tool maps a COBOL copybook into a Java class.

Synopsis

Invoke the tool with the following command:

```
java bea.jam.egen.EgenCobol scriptfile
```

where:

`java`

is the name of the Java virtual machine executable in the Java Development Kit (JDK).

`bea.jam.egen.EgenCobol`

is the full class name of the EgenCobol tool.

scriptfile

is the script file that controls the EgenCobol tool. You must write this script file on an application-by-application basis. (See Listing A-1 for an example).

Listing A-1 Example of `scriptfile.egen`

```
### example script
#

view demo.CustomDataView from emprec.cpy

service demoService accepts CustomDataView returns CustomDataView

page demoPage "Demo Page"
{
    view demo.CustomDataView

    buttons
    {
        "Try It" service(demoService) shows demoPage
    }
}

servlet demo.DemoServlet shows demoPage
```

Script Syntax Reserved Words

The reserved words shown in Table A-1 must be used as specified in the “Grammar” section.

Note: A reserved word can be used as an identifier if it is enclosed in either single or double quotation marks (refer to “General rules”).

Table A-1 Reserved Words

accepts	buttons	client	ejb	from	group
is	method	class	page	reset	returns
server	service	servlet	shows	view	

General rules

- The '#' character and all following characters on the same line are a comment. Use the '#' character to specify commented text.
- The character sequence " //" and all following characters on the same line are a comment. Use the " //" characters to specify commented text.
- The character sequence " /* " and all following characters until the occurrence of the sequence " */ " are a comment. Use the " /* " characters to specify commented text that extends beyond one line.
- White space (including newlines) is not significant. White space includes newlines, carriage returns, tabs, spaces, etc.
- Any sequence of letters, digits, underscores, or periods is a word.
- Any word that does not match a reserved word is an identifier.
- Any sequence of characters is treated as an identifier if it is enclosed in either single or double quotes. This allows the use of reserved words and sequences that contain spaces.

Grammar

The following grammar is described using a modified Backus-Naur Form (BNF) syntax, such as in many industry-standard software reference guides. It specifies a context-free grammar. Reserved words are shown in bold. Comments are in *italics* preceded by a dash (*—*).

```
script:
    definition | script definition

definition:
    viewdef | servicedef | servletdef | ejbdef | classdef |
    pagedef

viewdef:
    view viewname from copybook

servicedef:
    service servicename accepts fullViewname returns
    fullViewname

servletdef:
    servlet classname shows pagename

ejbdef:
    clientejb | serverejb

clientejb:
    client ejb classname ejbregistration { clientmethods }

serverejb:
    server ejb classname ejbregistration { servermethods }

classdef:
    client class classname { clientmethods }

pagedef:
    page pagename title { view viewname buttons { buttonlist } }

buttonlist:
    buttondef | buttonlist buttondef

buttondef:
    servicebutton | ejbbutton

clientmethodslist:
    clientmethoddef | clientmethodlist clientmethoddef

clientmethoddef:
    method methodname is servicename
```

servicebutton:
 buttonname **service** (servicename) **shows** pagename

ejbbutton:
 buttonname ejbmethod (fullViewname) **returns** fullViewname

viewname:
 classname

fullViewname:
 viewname | viewname [codepage]

copybook:
 identifier
 —An identifier that names a file containing a COBOL data definition.

servicename:
 identifier
 —An identifier that matches a resource definition in your `jcrmgw.cfg` file

pagename:
 identifier
 —An identifier that names a page definition.

codepage:
 identifier
 —The name of a codepage to be used for character translation to/from mainframe data formats. This must be a codepage supported by the JDK being used.

methodname:
 identifier
 —The name to be given to a generated Java method.

classname:
 identifier
 —An identifier that names a Java class, including any package name.

ejbregistration:
 identifier
 —The name that will be used to register the home interface for an EJB.

title:
 identifier
 —The title to be placed into the HTML generated for a page.

buttonname:
 identifier
 —A button name that will be used in the HTML generated for a page.

```
ejbmethod:  
    identifier  
    —An EJB classname and method specification that should look like this:  
    package.ejbclass.method  
    or  
   .ejbclass.method
```

Results of Running the Code Generator

- Each view definition (described in the “Programming Reference” section of this guide) causes the specified COBOL copybook to be parsed and a Java source file for the specified `DataView` class to be generated in the current directory.
- Each servlet definition causes a Java source file to be generated in the current directory for the specified class.
- Each client class definition causes a Java source file to be generated in the current directory for the specified class.
- Each EJB definition causes the generation of three Java source files and a deployment descriptor text file into the current directory. The names of the generated files are listed in Table A-2.

Table A-2 Generated Files for EJB Definitions

Name of File	Purpose
classnameHome.java	EJB Home Interface
classnameBean.java	EJB Implementation class
classname.java	EJB Remote Interface
classnameDD.txt	EJB Deployment Descriptor

B Configuration Checker Utility

The Java Communications Resource Manager Gateway (JCRMGW) is a Java application that manages sessions providing access into and out of the Java environment. It is configured using a text file named `jcrmgw.cfg` that resides in the WebLogic server directory. This configuration file is processed every time the gateway starts. Requests coming from the mainframe are mapped to an EJB that services the request while requests going to the mainframe are mapped to a mainframe program which can be executed using a CICS DPL. The JCRMGW also supports IMS operations.

When making changes to this file, you can verify the contents by invoking the gateway's configuration processor directly from the command line. This verification process allows you to discover and correct any errors prior to starting the gateway.

This topic consists of the following sub-topics:

- `bea.sna.jcrmgw.jcrmConfigurator`

This topic describes the configuration file checker utility.

- `GWBOOT`

This topic describes the `gwboot` startup class and its options.

bea.sna.jcrmgw.jcrmConfigurator

Java Communications Resource Manager Gateway configuration (`jcrmgw.cfg`) checker class.

Synopsis

`bea.sna.jcrmgw.jcrmConfigurator`

Description

The `bea.sna.jcrmgw.jcrmConfigurator` is used to check the `jcrmgw.cfg` file prior to starting the `jcrmgw`. It is recommended to place it into a script file and run it with standard output redirected to a file. The resulting output will be either diagnostic messages indicating syntax errors in the configuration file or a formatted listing of the definitions as they will be used by the gateway.

There are no options and the only file which can be processed is the `./jcrmgw.cfg` file in the current directory.

GWBOOT

JCRMGW bootstrap command. This `gwboot` startup class launches the gateway when the server is started.

Synopsis

```
bea.sna.jcrgmw.gwboot [[-r] [-t] [-u] <userid> ]
```

Description

`gwboot` is used in the `weblogic.properties` file to start an instance of the `jcrgmw` and optionally spawn a CRM to listen on the address specified in the `jcrgmw.cfg` file if required.

The following options can be specified:

`-r`

specifies a remote SNACRM and suppresses the spawning of a CRM process. The CRM is assumed to already be running and listening at the address specified in the `SNACRMADDR` entry of the `jcrgmw.cfg` file.

This would be required when the CRM is running on a machine other than the gateway, or if the CRM was started in its own window for purposes of testing a configuration.

`-t`

specifies tracing CRM. Turns on level 3 CRM tracing. This can only be used if the gateway is spawning a new CRM. If the CRM is running remotely and tracing is required, use the `-t` option on the command line which started the CRM.

`-u<userid>`

specifies `<userid>`. This is the default userid to be used by the gateway for all APPC requests. This is useful for IDENTITY type security when the gateway clients do not set a userid.

C Error and Informational Messages

This document contains the following descriptions of error and informational messages that can be encountered while using JAM.

The following table contains a description of error, informational, and warning messages that can be encountered while using the JAM software.

100	warning: 66 level (RENAMES) is not supported
DESCRIPTION	This language feature is not supported.
ACTION	No action is necessary.
101	warning: 88 level (condition name) is not supported
DESCRIPTION	This language feature is not supported.
ACTION	No action is necessary.
102	warning: Binary bitfield datatype is not supported
DESCRIPTION	This language feature is not supported.
ACTION	No action is necessary.

103	warning: COMP-5 datatype is not supported
DESCRIPTION	This language feature is not supported.
ACTION	No action is necessary, but it is recommended that the source file be corrected.
104	warning: COMP-X datatype is not supported
DESCRIPTION	This language feature is not supported.
ACTION	No action is necessary, but it is recommended that the source file be corrected.
105	warning: Extraneous ' ' ignored
DESCRIPTION	A extra delimiter was encountered, and is ignored.
ACTION	No action is necessary.
106	warning: Extraneous OCCURS TO clause, ignored
DESCRIPTION	This clause is not necessary, and is ignored.
ACTION	No action is necessary, but it is recommended that the source file be corrected.
107	warning: INDEXED BY clause ignored
DESCRIPTION	This clause is not necessary, and is ignored.
ACTION	No action is necessary.
108	warning: Identifier is not unique: {name}
DESCRIPTION	The data item name is not unique, which might cause ambiguity.
ACTION	No action is necessary.
109	warning: KEY IS clause ignored
DESCRIPTION	This clause is not necessary, and is ignored.
ACTION	No action is necessary.

110	warning: Level number {num} is out of sequence, assuming {num}	
	DESCRIPTION	The level number of a data item definition does not match previous level numbers, so a default value is assumed.
	ACTION	No action is necessary, but it is recommended that the source file be corrected.
111	warning: OCCURS lower bound exceeds upper bound ({occurMin} > {occurMax})	
	DESCRIPTION	The OCCURS ranges are out of order.
	ACTION	No action is necessary, but it is recommended that the source file be corrected.
112	warning: PICTURE ignored for COMP-1/COMP-2 datatype	
	DESCRIPTION	The clause is not meaningful for the data item definition.
	ACTION	No action is necessary.
113	warning: PICTURE ignored for INDEX datatype	
	DESCRIPTION	The clause is not meaningful for the data item definition.
	ACTION	No action is necessary, but it is recommended that the source file be corrected.
114	warning: PICTURE ignored for POINTER datatype	
	DESCRIPTION	The clause is not meaningful for the data item definition.
	ACTION	No action is necessary, but it is recommended that the source file be corrected.
115	warning: PICTURE ignored for binary bitfield datatype	
	DESCRIPTION	The clause is not meaningful for the data item definition.
	ACTION	No action is necessary, but it is recommended that the source file be corrected.

116	warning: POINTER datatype is not supported
DESCRIPTION	This language feature is not supported, and is treated as un-typed data.
ACTION	No action is necessary.
117	warning: PROCEDURE-POINTER datatype is not supported
DESCRIPTION	This language feature is not supported, and is treated as un-typed data.
ACTION	No action is necessary.
118	warning: Token begins with an unrecognizable character ({char})
DESCRIPTION	An unrecognizable character was encountered in the source file.
ACTION	No action is necessary, but it is recommended that the source file be corrected.
119	warning: USAGE ignored for 88-level datatype
DESCRIPTION	The clause is not meaningful for the data item definition.
ACTION	No action is necessary, but it is recommended that the source file be corrected.
200	error: BLANK WHEN ZERO is only valid for zoned numeric types
DESCRIPTION	The clause is not meaningful for the data item definition.
ACTION	Correct the source file.
201	error: Bad data item clause
DESCRIPTION	A syntax error or semantic disagreement was encountered while parsing the data item definition.
ACTION	Correct the source file.
202	error: Cannot REDEFINE the item: {name}
DESCRIPTION	The name specified is not a valid REDEFINES target.
ACTION	Correct the source file.

203	error: Character literal is missing its closing quote
DESCRIPTION	Quoted literals require a closing quote mark.
ACTION	Correct the source file.
204	error: Character literal is too long, truncated ({num})
DESCRIPTION	Character literals are truncated beyond a fixed upper limit.
ACTION	Correct the source file.
205	error: DEPENDING ON clause requires OCCURS TO upper bound
DESCRIPTION	The required clause is missing.
ACTION	Correct the source file.
206	error: DEPENDING ON item is not an integer: {name}
DESCRIPTION	The DEPENDING ON data item is not a numeric integer type.
ACTION	Correct the source file.
207	error: DEPENDING ON requires an OCCURS clause
DESCRIPTION	The required clause is missing.
ACTION	Correct the source file.
208	error: Expected ASCENDING/DESCENDING KEY IS clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
209	error: Expected BLANK
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.

210	error: Expected <code>DEPENDING ON</code> clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
211	error: Expected <code>DEPENDING ON</code> qualified identifier
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
212	error: Expected <code>EXTERNAL</code>
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
213	error: Expected <code>EXTERNAL/GLOBAL</code>
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
214	error: Expected <code>GLOBAL</code>
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
215	error: Expected <code>INDEXED BY</code> clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
216	error: Expected <code>INDEXED BY</code> qualified identifier
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.

217	error: Expected JUSTIFIED clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
218	error: Expected KEY IS qualified identifier
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
219	error: Expected LEADING/TRAILING, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
220	error: Expected OCCURS clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
221	error: Expected OCCURS lower bound, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
222	error: Expected OCCURS upper bound, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
223	error: Expected PICTURE clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.

224	error: Expected PICTURE specification, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
225	error: Expected REDEFINES clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
226	error: Expected REDEFINES identifier, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
227	error: Expected RENAMES THRU identifier, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
228	error: Expected RENAMES clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
229	error: Expected RENAMES qualified identifier, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
230	error: Expected SIGN clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.

231	error: Expected SYNC clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
232	error: Expected VALUE clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
233	error: Expected ZERO
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
234	error: Expected a ')' following a bitfield size, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
235	error: Expected a USAGE data type, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
236	error: Expected a bitfield size, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
237	error: Expected a data clause, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.

238	error: Expected a level number, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
239	error: Expected an identifier or FILLER, found '{text}'
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
240	error: Extraneous BLANK WHEN ZERO clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
241	error: Extraneous DEPENDING ON clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
242	error: Extraneous EXTERNAL clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
243	error: Extraneous GLOBAL clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
244	error: Extraneous INDEXED BY clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.

245	error: Extraneous JUSTIFY clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
246	error: Extraneous KEY IS clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
247	error: Extraneous OCCURS clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
248	error: Extraneous PICTURE clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
249	error: Extraneous REDEFINES clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
250	error: Extraneous RENAMES clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
251	error: Extraneous SIGN clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.

252	error: Extraneous SYNC clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
253	error: Extraneous USAGE clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
254	error: Extraneous VALUES clause
DESCRIPTION	The data item definition can only have one such clause.
ACTION	Correct the source file.
255	error: Hex string literal must have an even number of digits
DESCRIPTION	Hexadecimal character literals must be composed of an even number of digits.
ACTION	Correct the source file.
256	error: INDEXED BY requires an OCCURS clause
DESCRIPTION	The required clause is missing.
ACTION	Correct the source file.
257	error: Improper bitfield size ({len})
DESCRIPTION	A improper length was specified.
ACTION	Correct the source file.
258	error: Improper level-number for REDEFINES item ({levelNo})
DESCRIPTION	The level numbers of redefined data items must match.
ACTION	Correct the source file.

259	error: JUSTIFY is only valid for alphanumeric types
DESCRIPTION	The clause is not meaningful for the data item definition.
ACTION	Correct the source file.
260	error: KEY IS requires an OCCURS clause
DESCRIPTION	The required clause is missing.
ACTION	Correct the source file.
261	error: Level number {num} is out of sequence, assuming {num}
DESCRIPTION	The level number of a data item definition does not match previous level numbers, so a default value is assumed.
ACTION	Correct the source file.
262	error: Malformed DEPENDING ON clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
263	error: Malformed INDEXED BY clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
264	error: Malformed KEY IS clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
265	error: Malformed USAGE clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.

266	error: Malformed USAGE spec
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
267	error: Malformed VALUE clause
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
268	error: Malformed data definition ignored for: {name}
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
269	error: Malformed data definition, ignored
DESCRIPTION	A syntax error occurred while parsing the source file.
ACTION	Correct the source file.
270	error: Malformed picture specification: '{pic}'
DESCRIPTION	The picture clause contains invalid characters.
ACTION	Correct the source file.
271	error: Missing PICTURE clause
DESCRIPTION	The data item definition requires such a clause.
ACTION	Correct the source file.
272	error: Missing USAGE and PICTURE clauses
DESCRIPTION	The data item definition requires such a clause.
ACTION	Correct the source file.

273	error: Missing VALUE literal constant, found '{text}'
DESCRIPTION	The clause contains a syntax error.
ACTION	Correct the source file.
274	error: Missing identifier following IN/OF, found '{text}'
DESCRIPTION	The clause contains a syntax error.
ACTION	Correct the source file.
275	error: Missing literal constant after THRU, found '{text}'
DESCRIPTION	The clause contains a syntax error.
ACTION	Correct the source file.
276	error: Nonexistent or nonunique DEPENDING ON identifier: {name}
DESCRIPTION	The name specified is ambiguous.
ACTION	Correct the source file.
277	error: OCCURS count must be greater than zero ({occurMax})
DESCRIPTION	Arrays must have at least one element.
ACTION	Correct the source file.
278	error: REDEFINES identifier cannot be qualified
DESCRIPTION	The clause contains a syntax error.
ACTION	Correct the source file.
279	error: REDEFINES item must have the same level number ({levelNo})
DESCRIPTION	The level numbers of redefined data items must match.
ACTION	Correct the source file.
280	error: RENAMES is not supported
DESCRIPTION	This language feature is not supported.
ACTION	Correct the source file.

281	error: Recovering, skipping to next ' '
	DESCRIPTION A syntax error was encountered, so the rest of the definition is ignored.
	ACTION Correct the source file.
282	error: String literal is empty
	DESCRIPTION A quoted literal must contain at least one character.
	ACTION Correct the source file.
283	error: USAGE and PICTURE clauses disagree
	DESCRIPTION The clauses specify contradictory types or lengths.
	ACTION Correct the source file.
284	error: Word is too long, truncated ({num})
	DESCRIPTION Token words cannot be longer than a certain fixed length.
	ACTION Correct the source file.
300	Error: An I/O error occurred while generating [{name}]: {error}
	DESCRIPTION Could not write to the output file.
	ACTION Check the permissions of the output file.
301	Error: An I/O error occurred while generating [{name}]: {error}
	DESCRIPTION Could not write to the output file.
	ACTION Check the permissions of the output file.
302	Error: An I/O error occurred while generating [{name}]: {error}
	DESCRIPTION Could not write to the output file.
	ACTION Check the permissions of the output file.

303	Error: An I/O error occurred while generating [{view}]: {error}
	DESCRIPTION Could not write to the output file.
	ACTION Check the permissions of the output file.
304	Error: An I/O error occurred while reading the script: {file}
	DESCRIPTION The file could not be read.
	ACTION Check the permissions of the input file.
305	Error: EJB specification must contain both a class name and a method name
	DESCRIPTION Proper code cannot be generated without the missing items.
	ACTION Provide the missing items.
306	Error: EJB {bean} is not defined.
	DESCRIPTION A nonexistent EJB bean name was referenced.
	ACTION Specify a different file name.
307	Error: Parse failed on [{file}].
	DESCRIPTION A syntax error was encountered while parsing the script file.
	ACTION Correct the script.
308	Error: The copybook [{file}] was not found.
	DESCRIPTION A nonexistent COBOL source file name was specified.
	ACTION Correct the misspelling or provide the missing source file.
309	Error: The script file [{file}] was not found.
	DESCRIPTION A nonexistent file name was specified.
	ACTION Specify a different file name.

310	Error: excess method {name} is ignored.
	DESCRIPTION An extraneous method definition was specified.
	ACTION Remove the duplicate definition.
311	Error: expecting {token}.
	DESCRIPTION A syntax error occurred while parsing the input file.
	ACTION Correct the input file.
312	Error: method {name} is not defined in EJB {bean}.
	DESCRIPTION A nonexistent method was referenced.
	ACTION Correct the input file.
313	Error: service {name} is not defined.
	DESCRIPTION A nonexistent service name was referenced.
	ACTION Correct the input file.
314	Error: service {service} is not defined.
	DESCRIPTION The service name referenced was not defined.
	ACTION Provide the missing service name definition or correct the misspelling.
315	Error: servlet {name} refers to an unknown page ({page}).
	DESCRIPTION A nonexistent page name was referenced.
	ACTION Correct the input file.

D Java Docs

The BEA eLink Java Adapter for Mainframe WLS Edition (JAM) product comes with HTML pages that document the JAM Java classes. These also are referred to as “javadoc” files. They are located in the `jamdoc.jar` file, found in the JAM installation directory.

Issue the following command to extract the javadoc HTML files from the jar file:

```
jar -xvf jamdoc.jar
```

where:

`jar`

is the Java archive command.

`-xvf`

is the extract file(s) parameter.

`v`

is the verbose parameter to list the files.

`f`

is the option which designates the next parameter as the jar file name.

`jamdoc.jar`

is the name of the JAM javadoc file.

This command extracts all of the files contained in the jar file into the current directory. The HTML documentation files are placed in a newly created subdirectory named `classdocs` in the current directory.

To view an HTML documentation file, open your web browser and specify the file name of the javadoc you want to view, taken from the `classdocs` directory. Any of the following files are good for getting started:

- `classdocs/AllNames.html`
- `classdocs/packages.html`
- `classdocs/tree.html`

Glossary

A

Application Programming Interface (API)

1) The verbs and environment that exist at the application level to support a particular system software product. 2) A set of code that enables a developer to initiate and complete client/server requests within an application. 3) A set of calling conventions that define how to invoke a service. A set of well-defined programming interfaces (entry points, calling parameters, and return values) by which one software program utilizes the services of another

Application Program-to-Program Communication (APPC)

An interface to LU6.2 services; provides a set of primitives to conduct conversations in LU6.2 sessions.

B

buffer

see "typed data buffer."

C

CICS

"Customer Information Control System." A program execution monitor system resembling an operating system that controls and manages the execution of multiple transactional programs. CICS programs are arranged within "regions." COBOL is the primary programming language used for CICS applications.

COBOL

"Common Business Oriented Language." A semi-structured programming language designed by the U.S. DoD in 1959, designed to be used for most business applications. It is the most common language used for most mainframe applications in general.

COMMAREA

A CICS data buffer area that is sent to application programs, typically as a COBOL linkage section variable, providing an area for sending data to, and returning data from, application programs. This is part of the data sent to a program invoked via DPL.

copybook

A COBOL source file that is included by one or more source programs and typically contains a declaration for one or more record (group) data items.

D

DataView

The representation of a data record, consisting of information about all of the data fields within the record as well as their types, lengths, and offsets.

DPL

"Distributed Program Link." A remote invocation of a CICS program from another CICS region or SNA domain.

Domain

A *domain* can be another BEA WebLogic Server application that is independently administered, an application that is under the control of another transaction processing system, or an application in a remote CICS/ESA region. Domains can be local or remote.

E

EJB

"Enterprise Java Beans." The object-oriented business component model for transactional-based programs, defining the operation and protocols between Java client and server programs.

I

Information Management System (IMS)

A database manager used by CICS/ESA to allow access to data. IMS provides for the arrangement of data in an hierarchical structure and a common access approach in application programs that manipulate IMS databases.

J

J2EE

"Java to Enterprise Edition." A collection of Java standards (EJB, JNI, and so forth) that together define an enterprise application programming environment.

Java

An object-oriented programming language. A descendant from C, it was invented by James Gosling of Sun Microsystems.

L

Local Domain

A *Local Domain* is a part of an application (set or subset of services) that is available to other domains. A Local Domain is always represented by a Domain Gateway Group, and the terms are used interchangeably.

Local Service

A *Local Service* is a service of a local domain that is made available to remote domains through a Domain Gateway Group.

Logical Unit (LU)

In SNA, a port through which a user gains access to the services of a network. Also, see System Network Architecture (SNA).

LU6.2

LU6.2 is a particular SNA logical unit that identifies a specific set of services for program to program communication. Services include syncpoint, mapping of buffers into records, message confirmation, and security.

P

Partitioned Data Set (PDS).

A CICS/ESA data set in direct access storage that is divided into partitions called members. A member can contain a program or data. Program libraries are held in partitioned data sets.

R

re-entrant

The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

Remote Domain

A *Remote Domain* is a part of an application accessed through a Domain Gateway Group. The remote domain may be another BEA WebLogic Server application, an application running under another TP system, or a BEA JAM application.

Remote Service

A *Remote Service* is a service of a remote domain that is made available to the local application through a Domain Gateway Group.

S

Session

When two LU bind with each other, that is, when they have successfully negotiated how they will communicate, they are said to be in *session*. SNA has fixed limits on the number of sessions configured for an LU type.

Stack

Platform vendor-supplied software that provides connectivity to an SNA network.

SNA Communication Resource Manager (SNACRM)

A process that directly communicates with the PU2.1 server.

System Network Architecture (SNA)

A seven-layer networking protocol. Each layer of the protocol has a set of associated data communication services. The services of the uppermost layer are embodied in a Logical Unit (LU). Each LU type defined in SNA has its own specific set of services available to an end user for communicating. The end user may be a terminal device, or an application program. The SNA structure enables the end user to operate independently, unaffected by the specific facilities used for information exchange.

T

Transaction

- 1) A complete unit of work that transforms a database from one consistent state to another. In DTP, a transaction can include multiple units of work performed on one or more systems.
- 2) A logical construct through which applications perform work on shared resources (e.g., databases). The work done on behalf of the transaction conforms to the four ACID Properties: atomicity, consistency, isolation, and durability.

typed data buffer

A block of data, arranged as a record composed of one or more data fields. Each field has a type and a length, and optionally a name. Buffer types are specified by their name, which typically corresponds to the copybook name or class name that defines the record structure.

V

Virtual Telecommunications Access Method (VTAM)

A set of programs that control communication across a network between terminals and application programs.



Index

A

- accessors 6-4
- Adobe Acrobat Reader xiii
- alphanumeric field
 - rules for mapping 6-2
- APPC 1-1, 2-1
 - sign-on security style in JAM
 - applications 5-1
- Application Programming Interface (API)
 - platform 2-1
- array field
 - rules for mapping 6-5

B

- BigDecimal
 - rules for mapping to 6-3
- BLANK WHEN ZERO field
 - rules for mapping 6-2
- Browser session flowchart 3-22

C

- CICS
 - COMMAREA specified by COBOL
 - copybook 3-2
 - Link command 2-1
- CICS application 1-1
- CICS DPL 2-12
- COBOL copybook 1-4, 3-1
 - creating new 3-5

LINKAGE SECTION 3-5

- processing by EngenCobol code
 - generator A-6
- rules for mapping into a Java class 6-1
- rules for mapping REDEFINES 6-5
- using existing 3-5

COBOL data types

- syntax features and data types supported
 - by EngenCobol tool 6-6
- context-free grammar
 - rules for EngenCobol code generator
 - script A-4
- conversations between applications x

D

- Distributed Program Link (DPL) 1-1, 2-1

E

- edited numeric field
 - rules for mapping 6-2
- EGEN code generator 1-2
- EGEN translator 1-4
 - EngenCobol code generator A-1
 - generation models 1-4
- EngenCobol code generator
 - rules for generating code 6-1
 - rules for writing script file A-1
- EJB
 - deploying 4-3
 - deployment descriptor generated by

- EgenCobol code generator A-6
- Home Interface class generated by
 - EgenCobol code generator A-6
- hot-deployed 4-4
- Implementation class generated by
 - EgenCobol code generator A-6
- Remote Interface class generated by
 - EgenCobol code generator A-6
- EJB application
 - customizing 3-47
 - customizing Java classes in 3-34
 - deploying 4-1
 - generating as a remote server 3-41
 - producing Java classes for 3-26
 - sample script for defining 3-27
 - sample script process command 3-29
 - Stateless Session EJB 3-36
- elementary field
 - rules for mapping 6-4
- Enterprise Java Beans (EJB) 1-2

F

- field name
 - rules for mapping into Java name 6-2
- field type
 - rules for mapping into Java type 6-2
- FUNCTION
 - as optional keyword for remote services 2-14

G

- group field
 - nested, rules for mapping 6-3
 - rules for mapping 6-2
- gwboot startup class B-1

H

- Home Interface implementation class
 - building 4-3
- HTML
 - Java Docs E-1
 - Web User Interface for this guide xiii

I

- IMS 1-4, 2-1
- IMS application 1-1
- INDEX field
 - rules for mapping 6-3

J

- JAM configuration
 - combined 1-6
 - distributed 1-7
 - order for configuring the JAM environment 2-2
- JAM Java classes
 - documented in Java Docs E-1
- jar file
 - extracting Java Docs from E-1
 - jam_11.jar file on product CDROM 6-1
- Java
 - request/response 1-1
- Java application
 - customizing servlet-only application 3-17
 - customizing simple stand-alone application 3-55
 - generating a simple stand-alone application 3-51
 - generating servlet-only application 3-11
 - generating the base Java application 3-1
 - models 3-10
 - sample of generated source file 3-16
 - typed data buffer 3-2

Java clients
 send/receive data buffers with
 COBOL/CICS DPL programs
 3-2

Java data types
 converting to COBOL data types 3-5

Java Development Kit (JDK) A-1

Java servlets 1-2

JCRMGW 1-2, 1-3, 1-5
 configuration file for 2-6
 gwboot command B-3
 sample configuration file 2-15
 setting address of associated SNACRM
 2-8
 setting tracing for B-3
 using GROUP parameter to correlate
 with a SNACRM 2-8
 weblogic.properties file 2-3
 WEBLOGICCLASSPATH 2-3
jcrmgw.cfg file 2-7
 verifying with configuration checker
 utility B-2

JUSTIFIED RIGHT field
 rules for mapping 6-2

L

Local LU 2-2
 SNA stack parameter for 1-5

M

Mainframe Application Alias 2-7
Mainframe Applications
 mapping program names to EJB servers
 2-12
 mapping program names to method
 names 2-13
MAX sessions
 parameter 2-3
 SNA stack parameter 1-5

Minimum Contention Winners
 parameter 2-3
 SNA stack parameter 1-5
Mode name
 SNA stack parameter 1-5

N

numeric field
 rules for mapping 6-3

O

OS390/CICS programs ix

P

password
 security settings for 5-1
PU2.1 server (stack) 1-6

R

RDOM
 as keyword for remote services 2-13
REDEFINES clause
 rules for mapping 6-5
remote Interface implementation class
 building 4-3
Remote LU 2-2
 setting alias for SNACRM partner
 application 2-10
 SNA stack parameter 1-5
remote names x
RNAME
 as keyword for remote services 2-14

S

Script process command 3-15

Security

- RACF profile for applications 5-1
- settings 2-12

servlet

- accessing with URL 4-2
- deploying 4-1

SIGN IS LEADING field

- rules for mapping 6-2

SIGN IS TRAILING field

- rules for mapping 6-2

SNA

- maximum number of sessions per link 2-11
- setting minimum number of sessions as contention winners 2-11

SNA Mode Name 2-2

SNA protocol 1-6

SNA stack 1-2, 1-5

SNACRM 1-2, 1-7

- identifying partner mainframe application regions 2-10
- Local LU parameter 2-2
- setting address in associated JCRMGW 2-8
- specifying Local LU for 2-9
- specifying stack for 2-9
- turning on tracing for 2-4
- using GROUP parameter to correlate with a JCRMGW 2-8

stack configuration x

Supported SNA stacks 2-3

T

TCP/IP 1-7

Technical support xv

TRANTIME

- as optional keyword for remote services 2-14

TUXEDO application administrator x

U

userid

- security settings for 5-1
- setting in JCRMGW B-3

V

VTAM

- mode table 2-10

W

Web browser

- viewing Java Docs in E-2

Web User Interface (WUI)

- HTML interface for this guide xiii

WebLogic Express (WLX) 1-2

WebLogic Server (WLS) 1-1, 1-7

- as Java server 3-23
- deploying applications to WLS system 4-1
- editing properties file to add EJB 4-3
- in the JAM environment 1-2
- startup 2-3
- WEBLOGICCLASSPATH 3-61
- weblogics.properties file
- modifying to deploy servlet 4-2