

iWay

iWay Java Adapter for Mainframe Samples Guide

Version 5.1 Service Pack 1

EDA, EDA/SQL, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, FOCUS Vision, Hospital-Trac, Information Builders, the Information Builders logo, Parlay, PC/FOCUS, SmartMart, SmartMode, SNAPpack, TableTalk, WALDO, Web390, WebFOCUS and WorldMART are registered trademarks, and iWay and iWay Software are trademarks of Information Builders, Inc.

Due to the nature of this material, this document refers to numerous hardware and software products by their trademarks. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not this publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Copyright © 2007, by Information Builders, Inc and iWay Software. All rights reserved. Patent Pending. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Preface

iWay Java Adapter for Mainframe (iWay JAM) includes complete samples that are installed as part of the product installation. These samples are provided to demonstrate how iWay JAM integrates WebLogic and mainframe applications.

This documentation explains how to run the samples that are included with iWay JAM.

How This Manual Is Organized

The following table lists the titles and numbers of the chapters and the appendix for this manual with a brief description of the contents of each chapter or appendix.

Chapter		Contents
1	iWay Java Adapter for Mainframe Samples Overview	Provides an overview of the samples that are included with iWay JAM.
2	Using the IMS Samples	Demonstrate how iWay Java Adapter for Mainframe (iWay JAM) integrates WebLogic applications with IMS applications on a mainframe.
3	Using the CICS Samples	Demonstrate how iWay Java Adapter for Mainframe (iWay JAM) integrates the WebLogic applications with CICS applications.
4	Using the Explicit APPC Samples	Demonstrates how iWay Java Adapter for Mainframe (iWay JAM) integrates the WebLogic applications with batch MVS COBOL applications.

Documentation Conventions

The following table lists and describes the conventions that apply throughout this manual.

Convention	Description
<code>THIS TYPEFACE</code> or <code>this typeface</code>	Denotes syntax that you must enter exactly as shown.
<code>this typeface</code>	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
<u><code>underscore</code></u>	Indicates a default setting.
<i>this typeface</i>	Represents a placeholder (or variable) in a text paragraph, a cross-reference, or an important term.
this typeface	Highlights a file name or command in a text paragraph that must be lowercase.
<i>this typeface</i>	Indicates a button, menu item, or dialog box option you can click or select.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices; type one of them, not the braces.
	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis points (...).
.	Indicates that there are (or could be) intervening or additional commands.

Help Us to Serve You Better

To help our consultants answer your questions effectively, please be prepared to provide specifications and sample files and to answer questions about errors and problems.

The following tables list the specifications our consultants require.

WebLogic Server Platform Operating System and Operating System Version	
WebLogic Server Version and Service Pack / special patch	
iWay JAM CRM Platform Operating System	
CICS or IMS Operating System and Operating System Version	
CICS or IMS Version Information	
Third party SNA stack product and version	

The following table lists components. Specify the version in the column provided.

Component	
iWay JAM Gateway Build / Fix Level	
iWay JAM CRM Build / Fix Level	
Pre-existing eGen Application Compiles? Specify Yes or No.	

In the following table, specify the JVM version and vendor in the columns provided.

Version	Vendor

The following table lists additional questions to help us serve you better.

Request/Question	Error/Problem Details or Information
Provide usage scenarios or summarize the application that produces the problem.	
Did this happen previously?	
Is this configuration working on any other system?	
Can you reproduce this problem consistently?	
Any change in the application environment including: • Migrating to a new WebLogic Server service pack or version. • Installing a new operating system on WebLogic Server side or CRM component side. • Migrating to a new CICS or IMS region, version, or operating system.	
Under what circumstance does the problem <i>not</i> occur?	
Describe the steps to reproduce the problem.	
Describe the problem .	

Request/Question	Error/Problem Details or Information
Specify the error message(s).	

The following table lists error/problem files that might be applicable.

Error/Problem Files	Error/Problem File Detail or Information
WebLogic Server Application logs or error messages	
CICS or IMS application logs or error messages	
WebLogic Server configuration file	
iWay JAM configuration file	
Third party stack SNA configuration	
VTAM Logical Unit definitions	
iWay JAM CRM startup script and script messages	
iWay JAM CRM JCL and JOB information (JESMSG LG, JESJCL, JEYSYSMSG)	
iWay JAM gateway trace diagnostics	
iWay JAM CRM trace diagnostics	
APPC trace files	

If you are requested to collect additional iWay JAM diagnostic files, see the Diagnostics section in the *iWay Java Adapter for Mainframe Programming Guide*.

User Feedback

In an effort to produce effective documentation, the Documentation Services staff welcomes your opinions regarding this manual. Please use the Reader Comments form at the end of this manual to communicate suggestions for improving this publication or to alert us to corrections. You also can go to our Web site, <http://www.iwaysoftware.com> and use the Documentation Feedback form.

Thank you, in advance, for your comments.

iWay Software Training and Professional Services

Interested in training? Our Education Department offers a wide variety of training courses for iWay Software and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site, <http://www.iwaysoftware.com> or call (800) 969-INFO to speak to an Education Representative.

Interested in technical assistance for your implementation? Our Professional Services department provides expert design, systems architecture, implementation, and project management services for all your business integration projects. For information, visit our World Wide Web site, <http://www.iwaysoftware.com>.

Contents

1. iWay Java Adapter for Mainframe Samples Overview	1-1
About the iWay JAM Samples	1-2
Verification Samples	1-2
Programming Samples	1-2
About the Samples User	1-3
How to Use the Samples	1-4
Java Sample Code	1-5
Mainframe Sample Code	1-5
Preconfigured WebLogic Server Domains	1-5
Before You Run the Samples	1-6
2. Using the IMS Samples	2-1
About the IMS Samples	2-2
IMS Application to WebLogic Server JMS Topic	2-2
Java Client to IMS Sample Application	2-2
Transactional Sample from WebLogic Server to IMS	2-3
Transactional Sample IMS to WebLogic Server	2-3
Roadmap for the Samples	2-4
Using the Samples	2-4
Preparing to Use the IMS Samples	2-4
Using the IMS Application to WebLogic Server JMS Topic	2-11
WebLogic Application	2-12
IMS Program	2-12
Using the Java Client to IMS Sample Application	2-16
WebLogic Application	2-17
IMS Program	2-18
Running the Sample	2-20
Using the Transactional Sample from WebLogic Server to IMS	2-21
WebLogic Application	2-21
IMS Program	2-22
Running the Sample	2-25
Using the Transactional IMS to WebLogic Server	2-25
3. Using the CICS Samples	3-1
About the CICS Samples	3-2
CICS Application to WebLogic Server Sample EJB	3-2
Java Client to CICS Sample Application	3-2
Transactional Sample from WebLogic Server to CICS	3-2
Roadmap for the Samples	3-3

Using the Samples	3-3
Preparing to Use the CICS Samples	3-3
Using the CICS Application to WebLogic Server Sample EJB	3-10
WebLogic Application	3-11
CICS Program	3-11
Setting Up the Sample	3-13
Running the Sample	3-16
Using the Java Client to CICS Sample Application	3-16
WebLogic Application	3-16
CICS Programs	3-17
Setting Up the Sample	3-19
Running the Sample	3-22
Using the Transactional Sample from WebLogic Server to CICS	3-22
WebLogic Application	3-23
CICS Programs	3-24
Setting Up the Sample	3-25
Running the Sample	3-29
4. Using the Explicit APPC Samples	4-1
About the Explicit APPC Samples	4-2
Batch MVS COBOL Client to WebLogic EJB Sample	4-2
Transactional Batch MVS COBOL Client to WebLogic EJB Sample	4-2
Roadmap for the Samples	4-2
Using the Samples	4-3
Preparing to Use the Explicit APPC Sample	4-3
Using the Batch MVS COBOL Client to WebLogic EJB Sample	4-11
WebLogic Application	4-12
Setting Up the Sample	4-14
Using the Transactional Batch MVS COBOL Client to WebLogic EJB Sample	4-17

CHAPTER 1

iWay Java Adapter for Mainframe Samples Overview

Topics:

- About the iWay JAM Samples
- About the Samples User
- How to Use the Samples
- Before You Run the Samples

iWay Java Adapter for Mainframe (iWay JAM) is designed to integrate applications running in a WebLogic Server environment with applications running in a mainframe environment.

iWay JAM includes complete samples that are installed as part of the product installation. These samples are provided to demonstrate how iWay JAM integrates WebLogic and mainframe applications. The following topic provides an overview of the samples that are included with iWay JAM.

About the iWay JAM Samples

The samples included with iWay JAM enable you to see different capabilities of iWay JAM. The samples are much less complex than any standard production application and are not intended to be guides to application programming.

Samples have been included as a part of the iWay JAM software to:

- Verify that the product was installed correctly
- Demonstrate the different ways iWay JAM integrates WebLogic and mainframe applications

Samples are designed to run "out of the box" with minimal modification. Most tasks are preconfigured or configured during installation by the installer program. You are required to only enter configuration that applies to your specific environment.

Two types of samples are installed when you install iWay JAM:

- Verification Samples
- Programming Samples

Verification Samples

iWay JAM provides two installation verification samples, one for an IMS region and one for a CICS region. These samples allow you to verify the success of the iWay JAM installation as well as demonstrate iWay JAM integration in a WebLogic and mainframe environment. These samples are described in the *iWay Java Adapter for Mainframe Installation Guide*.

Programming Samples

The samples that are categorized as programming samples demonstrate how iWay JAM can be used with your system.

- IMS Samples
 - IMS Application to WebLogic Server Sample JMS Topic

This sample demonstrates making an asynchronous call through iWay JAM from IMS to insert a record into a JMS Topic running in WebLogic Server.
 - Java Client to IMS Sample Application

This sample demonstrates a Java client calling remote services located in an IMS region.
 - Transactional Sample from WebLogic Server to IMS

This sample demonstrates how a Java client makes calls to remote services located in an IMS region. The service calls that alter the data on the mainframe occur within the boundaries of two-phase commit transactions.

- Transactional Sample IMS to WebLogic Server EJB

This sample demonstrates an IMS call to an EJB method under the scope of an IMS transaction.

- CICS Samples

- CICS Application to WebLogic Server Sample EJBs

This sample demonstrates the functional capability provided by iWay JAM to invoke the services of an EJB (Enterprise Java Bean) deployed in a WebLogic Server from a CICS client.

- Java Client to CICS Sample Application

This sample demonstrates the invoking of CICS services from requests that originate from a Java client.

- Transactional Sample from WebLogic Server to CICS

This sample demonstrates making calls to remote services located in a CICS region from a Java client. The service calls that alter data on the mainframe occur within the boundaries of two-phase commit transactions.

- Explicit APPC Samples

- Batch MVS COBOL Client to WebLogic EJB Sample

The purpose of this sample is to demonstrate the functional capability of invoking the services of an EJB (Enterprise Java Bean) deployed in a WebLogic Server from the mainframe environment, specifically from a batch MVS client using explicit APPC.

- Transactional Batch MVS COBOL to WebLogic EJB Sample

The purpose of this sample is to demonstrate the functional capability of invoking the services of an EJB (Enterprise Java Bean) deployed in a WebLogic Server from the mainframe environment, under the scope of an RRS transaction.

About the Samples User

The samples are provided to demonstrate various capabilities and functions of iWay JAM. Samples users typically fulfill one of the following roles:

- Systems Programmer
- Mainframe Application Programmer

- Java Application Programmer

How to Use the Samples

All of the components necessary to run the examples documented in this manual, are installed with your iWay JAM product. You will find a working WebLogic domain with configuration files and pre-generated examples.

This is located in the following directory:

[iWay51/etc/samples/iwjam/domains/examples](#)

In addition, all of the source referenced in this document is installed in the following directory:

[iWay51/tools/egen/samples/examples](#)

While the iWay JAM samples contain precompiled source for the Java portion of the sample, you may choose to generate and compile source. Depending on your skill level, experience, and time-constraints, you may choose to work with the Java portion of the samples in the following ways:

- Run the sample

Because of the contents shipped with the samples, in most cases, you can simply run the sample with minimal configuration to run on your system. The sample will use the supplied .class files.
- Compile source and run the sample

Each sample provides build scripts that allow you to compile the source. This option allows you to see how the source is compiled and run the sample.
- Generate source, compile, and run the sample

Each sample provides eGen scripts. This option allows you to use the eGen Application Code Generator to generate the source. You can then run the build scripts. By using this option, you can see how the eGen utility generates source and run the sample.

In general, each of the samples include:

- Java Files
 - Java .class files
 - Java source
 - eGen scripts
 - Build scripts
 - Preconfigured WebLogic Server domain

- Mainframe Files
 - Mainframe source
 - Sample JCL
 - IMS or CICS sample configurations

Java Sample Code

The philosophy of keeping the samples simple is demonstrated in the Java segments of the samples. For samples with Java clients and mainframe servers, stand-alone client classes are used instead of using client EJBs because the standalone clients are simpler to code, understand, deploy, and include in your applications. The servlets that can be generated by the eGen Application Code Generator are not used because you may have difficulty enhancing them to improve their presentation or functionality.

Mainframe Sample Code

Mainframe sample programs are minimal, but are sophisticated enough to demonstrate the features of iWay JAM. Mainframe samples are designed to take advantage of the "least common denominator" of IMS or CICS features that you might have available. For example, some of the IMS samples interact with the installation verification transaction, IVTNO, that is shipped with IMS. CICS samples use Temporary Storage (TS) queues and VSAM files to imitate the behavior of databases, rather than assume you have a database installed.

Note: JCL to compile the mainframe sample programs is included as examples to provide completeness. The sample JCL may not conform to your standards or installations. It may need to be modified or replaced to meet your needs.

Program definitions and other configuration for your CICS or IMS region may require coordination with your IMS or CICS system programmer.

Preconfigured WebLogic Server Domains

The iWay JAM installation includes a directory named domains. This directory contains subdirectories for two preconfigured WebLogic Server domains. These WebLogic Server domains are:

- verify
WebLogic Server domain that is set up for running the installation verification
- examples
WebLogic Server domain that is set up for examples of iWay JAM running with WebLogic Server and mainframe applications

Before You Run the Samples

Before you run the samples, the following tasks must be completed:

1. Install WebLogic Server.

For information about installing WebLogic Server, see the BEA WebLogic Server documentation.

2. Install iWay JAM.

For information about installing iWay JAM, see the *iWay Java Adapter for Mainframe Installation Guide*.

3. Define the Logical Unit (LU) for the CRM and vary it active.

For information about defining the Logical Unit for the CRM, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*.

4. If using IMS, verify that the APPC communication to IMS is active.

For information about APPC communication with iWay JAM, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*.

5. If using CICS, the connection must be defined to the CICS region.

For information about CICS connection when using iWay JAM, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*.

After these tasks have been completed, determine which of the samples you want to run and see the corresponding section:

- Chapter 2, *Using the IMS Samples*
- Chapter 3, *Using the CICS Samples*
- Chapter 4, *Using the Explicit APPC Samples*

CHAPTER 2

Using the IMS Samples

Topics:

- About the IMS Samples
- Roadmap for the Samples
- Using the Samples

The IMS samples demonstrate how iWay Java Adapter for Mainframe (iWay JAM) integrates WebLogic applications with IMS applications on a mainframe.

About the IMS Samples

The following section provides a brief overview of each of the IMS samples described in this guide. For a detailed description of how each sample works and instructions for running each sample, see *Using the Samples* on page 2-4.

IMS Application to WebLogic Server JMS Topic

This sample demonstrates an asynchronous call through iWay JAM from IMS to a Java application running under WebLogic Server. In this sample, the supplied IMS client uses implicit APPC to make a request of a service advertised by the iWay JAM Gateway. This service takes all request data and places it on a JMS topic. The Gateway uses a DataView to convert the request data to XML before it is placed on the JMS topic. A topic receiver class is shipped with this sample so you can view the messages as they are placed on the JMS topic.

This sample highlights iWay JAM support of JMS that allows mainframe client programs to insert messages onto JMS queues or topics. In this sample, a COBOL copybook is created to use with the eGen Application Code Generator by examining the record layout for the IMS client application.

The iWay JAM Gateway uses DataView classes to translate the data received from the mainframe into XML. To accomplish this translation, the Gateway must be able to load the necessary DataView class; the necessary DataView class must be in the CLASSPATH set in the WebLogic Server startup script.

Java Client to IMS Sample Application

This sample demonstrates a Java client calling remote services located in an IMS application. The Java client receives a command and a record name from you. You enter one of the following commands: add, display, update, or delete. You may enter a host address and port if the gateway is running on a different machine. Depending on the command, the client may prompt you for additional information. The client then makes a service call to the installation verification transaction, IVTNO, that is shipped with IMS. The result displays.

IVTNO was chosen for a back-end application to this sample because IMS users will already have it installed or can easily do so. If you want to run this sample and IVTNO is not installed in your IMS region, coordinate the installation with your IMS system programmer. The use of IVTNO as a back-end application also illustrates how iWay JAM facilitates the integration of Java application with legacy applications without change to the legacy application. In this sample, COBOL copybooks are created to use with the eGen Application Code Generator by examining the record layout for the IMS application instead of using pre-existing copybooks. The record definitions are in the IMS sample program, DFSIVA1.

Transactional Sample from WebLogic Server to IMS

This sample demonstrates how a Java client makes calls to remote services located in an IMS application. The service calls that alter the data on the mainframe occur within the boundaries of two-phase commit transactions. The sample contains a transaction that is distributed over resources managed by WebLogic Server and resources located on the mainframe. This transaction uses a service call to add a record to the IVTNO database. The key to the record is inserted on a JMS queue within the boundaries of the same transaction as the service call to create the record. The queuing of the record key and the creation of the record in IMS will either be committed or rolled back together depending on the command line option you set.

The installation verification transaction, IVTNO, (shipped with IMS) was chosen for a back-end application to this sample because IMS users will already have it installed or can easily do so. If you want to run this sample and IVTNO is not installed in your IMS region, coordinate the installation with your IMS system programmer. The use of IVTNO also illustrates how iWay JAM facilitates the integration of Java applications with legacy applications with no change to the legacy application. In this sample, COBOL copybooks are created to use with the eGen utility by examining the record layout for the IMS application instead of using pre-existing copybooks. The record definitions are in the IMS sample program DFSIVA1.

Transactional Sample IMS to WebLogic Server

This sample demonstrates a synchronous call through iWay JAM from IMS to an EJB running in WebLogic Server. The call is scoped under an IMS transaction and the IMS program used is written to be "CPI-C driven". The CPI-C driven paradigm will guarantee that IMS resources and the protected conversation (with the CRM) are managed by the same IMS/RRS transaction. For more information, see the IBM documentation called *IMS Application Programming: Transaction Manager*.

A typical IMS program is converted to be CPI-C driven by accepting input and delivering output via APPC, and accessing IMS resources through the AIBTDLI interface.

The sample uses a batch program to invoke the CPI-C driven program. In practice, a front-end program may (via the IOPCB) drive the CPI-C driven program. The RRS calls are collected within a "C" interface, APPCCRM, included with the sample, to demonstrate a possible way to organize these calls in a reusable fashion.

The EJB method TOUPPER is invoked, as in the ExplicitAppc samples. This EJB method does no transactional work; it may be adapted to use transactional resources if desired.

This sample is for reference only. Full end-to-end deployment details are not yet available.

Roadmap for the Samples

To run the IMS samples, follow the roadmap listed below. General tasks for all of the IMS samples include:

1. Verify prerequisite tasks.

For a listing of prerequisite tasks, see *Before You Run the Samples* in Chapter 1, *iWay Java Adapter for Mainframe Samples Overview*.

2. Prepare to use the IMS sample.

- a. Start the CRM.
- b. Update the iWay JAM configuration file.
- c. Start the examples domain.
- d. Configure the iWay JAM Gateway.

Specific tasks for each sample include:

1. Set up the sample.

- a. Enable services.
- b. Set the environment.
- c. Generate and build source (optional).
- d. Complete mainframe tasks.

2. Run the sample.

Using the Samples

After you have completed the tasks described in *Before You Run the Samples* in Chapter 1, *iWay Java Adapter for Mainframe Samples Overview*, you are ready to use the sample.

Preparing to Use the IMS Samples

The following steps are common to all the IMS samples. These steps only need to be performed once for all IMS samples.

Step 1: Start the CRM

Before starting the iWay JAM Gateway, start the CRM. The CRM must be configured with certain parameter values at startup. These parameter values include:

- The address of the machine on which the CRM is running
- The port on which the CRM listens

- The name the Gateway will use to refer to the CRM

For running the samples, you must set the machine address and port. The values that you set for the machine address and port when the CRM is started, must agree with the values that you set for the CRM in the WebLogic Administration Console for the samples CRM. The name of the CRM that is preconfigured for running all of the samples is CRM1. Use this name when the CRM is started to run any of the samples.

The way you start the CRM depends on whether the CRM will be started under a Unix or MVS system. On Unix, start the CRM using a shell script. On MVS, start the CRM using JCL. For more information, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*.

Step 2: Update the iWay JAM Configuration File

On the machine where the Gateway is installed, switch to your examples domain by copying `jamconfig_IMS.xml` to `jamconfig.xml`.

Step 3: Start the examples Domain

From the command prompt, execute the following command from the examples directory to start the examples domain:

- For Microsoft Windows:

```
startExamplesServer.cmd
```

- For Unix:

```
./startExamplesServer.sh
```

Step 4: Configure the iWay JAM Gateway

Most configuration tasks are preconfigured or completed during the installation process by the installer program. For additional information about configuring iWay JAM, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*. However, you must make the following configuration changes for the IMS samples to run on your system.

Procedure: How to Configure the iWay JAM Gateway

To make configuration changes in the WebLogic Administration Console:

1. From your browser, open the WebLogic Administration Console using the following address:

```
http://hostname:7001/console
```

where:

```
hostname
```

Is the address of the machine where WebLogic Server is running.

7001

Is the port for WebLogic Server that has been configured for the examples domain.

2. When prompted, supply the following user and password information:

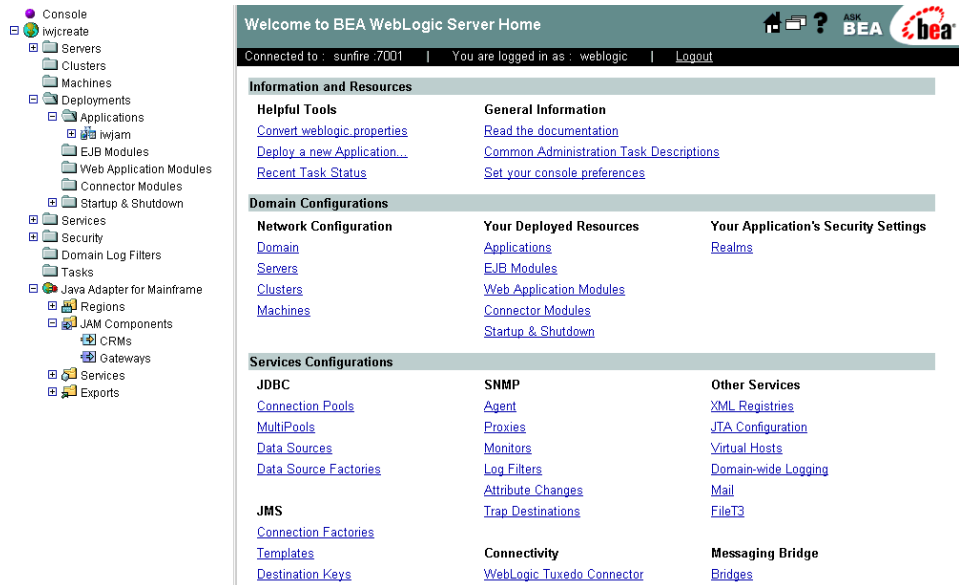
user: system

This user name cannot be changed.

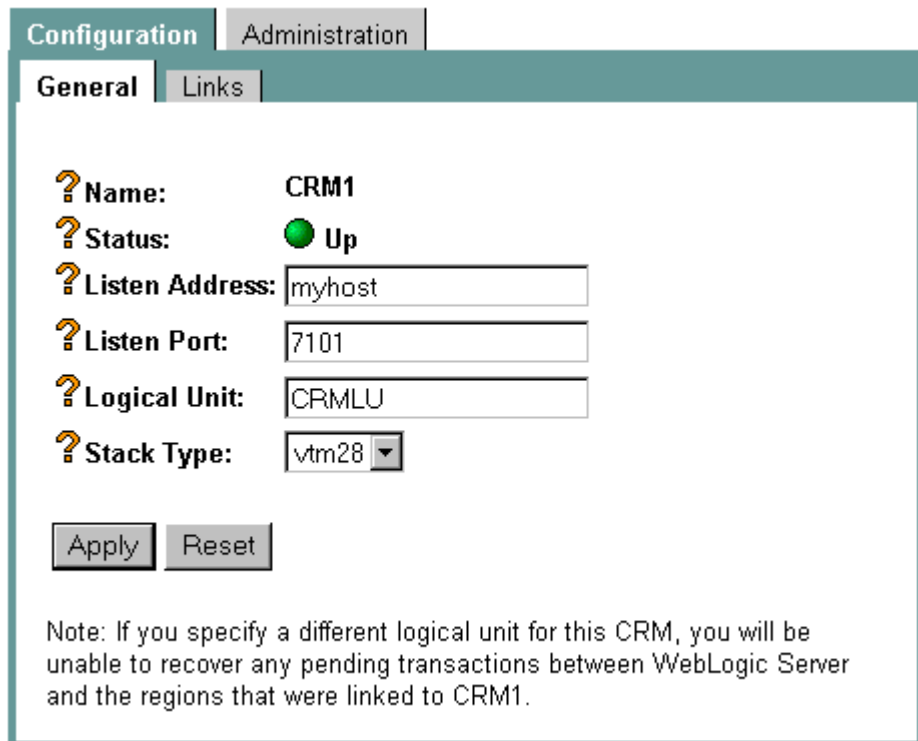
password: security

To change the password, see the BEA WebLogic Server documentation.

The WebLogic Administration Console appears.



3. To configure the CRM to the iWay JAM Gateway, in the left pane click *Java Adapter for Mainframe*, *JAM Components*, and then *CRMs*.
4. In the right pane, click *CRM1*.



Configuration Administration

General Links

? **Name:** CRM1

? **Status:**  Up

? **Listen Address:** myhost

? **Listen Port:** 7101

? **Logical Unit:** CRMLU

? **Stack Type:** vtm28 ▼

Apply Reset

Note: If you specify a different logical unit for this CRM, you will be unable to recover any pending transactions between WebLogic Server and the regions that were linked to CRM1.

- On the General tab, set the following fields to correspond with your system.

Field	Field Description
Listen Address	The address of the machine where the CRM is installed and running. This address must match the address set in the CRM startup JCL or script.
Listen Port	The port for the CRM. This entry must match the port set in the CRM startup JCL or script.
Logical Unit	The name of the Logical Unit defined for the CRM.
Stack Type	The stack type.

- Click *Apply*.

When the CRM is active, Status changes from red to green.

7. Configure the IMS region:

- a.** In the left pane, click *Java Adapter for Mainframe, Regions*, and then *IMS Regions*.
- b.** In the right pane, click *VS10IMS*.
- c.** Enter the Logical Unit name that supplies your IMS control region with APPC communication. You will find this name with the DISPLAY APPC operator command in IMS. Do not enter the Logical Unit name for APPLID of the IMS control region.

This APPLID does not support APPC communication. If your IMS control region does not currently support APPC communication, you will need to set up this communication in APPC/MVS. Then activate the communication within IMS using the START APPC Operator command.

- d.** Click *Apply* to set the Logical Unit.

This Logical Unit is not the same as the Logical Unit for the CRM in (3a).

The screenshot shows a 'Configuration' window with a 'General' tab. It contains two fields: 'Name' with the value 'VS10IMS' and 'Logical Unit' with the value 'IMSLU'. Below these fields are 'Apply' and 'Reset' buttons. A note at the bottom states: 'Note: If you specify a different logical unit for region VS10IMS, you will be unable to recover any transactions that were pending between WebLogic Server and this region.'

- e.** Click *IMS Regions* at the top of the right pane, and click *CRM1toIMS* in the new window.
- f.** On the Links tab, check *Deployed* and click *Apply*.

The screenshot shows a web-based configuration interface with two tabs: 'Configuration' and 'Administration'. The 'Administration' tab is active, and within it, the 'Links' sub-tab is selected. The main content area displays configuration details for a link named 'CRM1toIMS'. Each field is preceded by a question mark icon. The fields and their values are: Name (CRM1toIMS), Status (Down, indicated by a red circle icon), Region Name (VS10IMS (IMSRegion) in a dropdown), Logmode Name (SMSNA100 in a text box), Maximum Sessions (4 in a text box), Minimum Winner Sessions (2 in a text box), Security Level (Local in a dropdown), Default User-Id (empty text box), and Deployed (checked checkbox). Below these fields are 'Apply' and 'Reset' buttons. A note at the bottom states: 'Note: By modifying the parameters of this link you will be unable to recover any transactions that were pending on this link between WebLogic Server and region VS10IMS.'

? Name:	CRM1toIMS
? Status:	Down
? Region Name:	VS10IMS (IMSRegion) ▼
? Logmode Name:	SMSNA100
? Maximum Sessions:	4
? Minimum Winner Sessions:	2
? Security Level:	Local ▼
? Default User-Id:	
? Deployed:	☒

Note: By modifying the parameters of this link you will be unable to recover any transactions that were pending on this link between WebLogic Server and region VS10IMS.

- g. Click *Gateways* in the left pane, and *JAM5.1* in the right pane.

- h. On the General tab, check *Deployed* and click *Apply*.

The screenshot shows the 'Configuration' tab selected, with the 'General' sub-tab active. The gateway name is 'JAM5.1'. The status is 'Down' with a red circle icon. The 'Target WebLogic Server' is 'exampleserver' and the 'Target CRM' is 'CRM1'. The 'Deployed' checkbox is unchecked. There are 'Apply' and 'Reset' buttons. A note at the bottom states: 'Note: By modifying the Target CRM parameter of this gateway you will be unable to recover any transactions that were pending for gateway JAM5.1.'

8. To start the Gateway, select the *Administration* tab, *Start/Stop*, and then *Start*.

The screenshot shows the 'Administration' tab selected, with the 'Start / Stop' sub-tab active. The 'Monitor Gateway' section shows the name 'JAM5.1' and status 'Down' with a red circle icon. Below this, it says 'Gateway JAM5.1 is not available.'

If the Gateway is running, Status changes to green in the WebLogic Administration Console and the following message appears in the WebLogic Server log:

"JAM Gateway ready for use. Current link status: up(1)."

You have completed the general steps required to prepare your system to run the IMS samples. Select the IMS sample you want to run and follow the steps in that section to set up and run that sample.

Using the IMS Application to WebLogic Server JMS Topic

After completing the steps in the *Preparing to Use the IMS Samples* on page 2-4, you are ready to configure and run the IMS Sample Application to WebLogic Server JMS topic.

How the Sample Works

This sample illustrates making an asynchronous call through iWay JAM from an IMS client to a Java application running under WebLogic Server. In this sample, the supplied IMS client uses implicit APPC to make a request of a service advertised by the iWay JAM Gateway. This service takes all request data and places it on a JMS topic. The Gateway uses a DataView to convert the request data to XML before it is placed on the JMS topic.

Understanding the Sample Configuration

The client IMS program, IMSTOJMS, is defined to the IMS region in the same way any program is defined to an IMS region. No special considerations are required for use as a client making requests through iWay JAM.

The message inserted by IMSTOJMS on the IMS message queue is sent to iWay JAM. To accomplish this task, an IMS LU 6.2 descriptor must be created that maps the LTERM name that is passed to the program IMSTOJMS to the Logical Unit defined for the CRM and a transaction name. In this sample, the LTERM name is JAMIMS01. This is mapped in the sample IMS LU 6.2 descriptor DFS62DTI to the Logical Unit CRMLU. The transaction name is ITOJMSSV.

The CRMLU must be changed to the Logical Unit that was defined for the CRM. ITOJMSSV is the name of a JMSEvent in the iWay JAM configuration. The attributes of the ITOJMSSV JMSEvent in the iWay JAM configuration describe the JMS topic where the message will be queued and give the name of the DataView used to translate the message to XML. In this sample, the DataView is named Chardata.

Because the iWay JAM Gateway uses the Chardata DataView to translate the mainframe data to XML before placing it on the JMS topic, the Chardata.class file must be in the WebLogic Server CLASSPATH. The dataview directory is in the CLASSPATH set in the WebLogic Server startup script for the examples domain. The SUPPORTS XML directive is also included in the definition of the chardata.java in the eGen script chardata.egen.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Two classes compose the WebLogic side of this sample application:

- Chardata
- TopicReceive

Chardata is a DataView class that is generated by the eGen Application Code Generator. The data member in the Chardata.class corresponds to the data field in the CHARDATA copybook. The Chardata.class is responsible for all data translation between the mainframe format of the data and the Java format of the data. The iWay JAM Gateway uses this class to translate the mainframe data to XML.

The TopicReceive class allows you to view the messages as they are placed on the JMS topic. TopicReceive should be started before running the IMS client. TopicReceive establishes a connection to the JMS topic, receives messages placed on the topic by iWay JAM that have been sent by the IMS client, and then reports the messages to you. TopicReceive shuts down when a "quit" message is sent.

IMS Program

IMSTOJMS is a simple COBOL IMS client program that makes an asynchronous, no-response request by placing the request on the IMS message queue. IMSTOJMS receives an LTERM name and a string input from you. It issues a change call to change its output destination to the specified LTERM. Then it inserts the input string to this destination. Because the IMS LU 6.2 descriptor has been created and associated with the LTERM name, the message is sent to the CRM and then on to the iWay JAM Gateway. No special considerations are required in this program as a result of being used as a client making requests of a Java server through iWay JAM.

Example: Sample Files

The following table lists the sample files and their purpose:

File Name	File Purpose
chardata.cpy	COBOL copybook that defines the structure of the string mainframe data.
chardata.egen	eGen script that generates the Chardata.java DataView class.
Chardata.java	DataView class that corresponds to the chardata.cpy COBOL copybook.
TopicReceive.java	Class that implements MessageListener interface used to monitor the JMS topic for incoming messages.

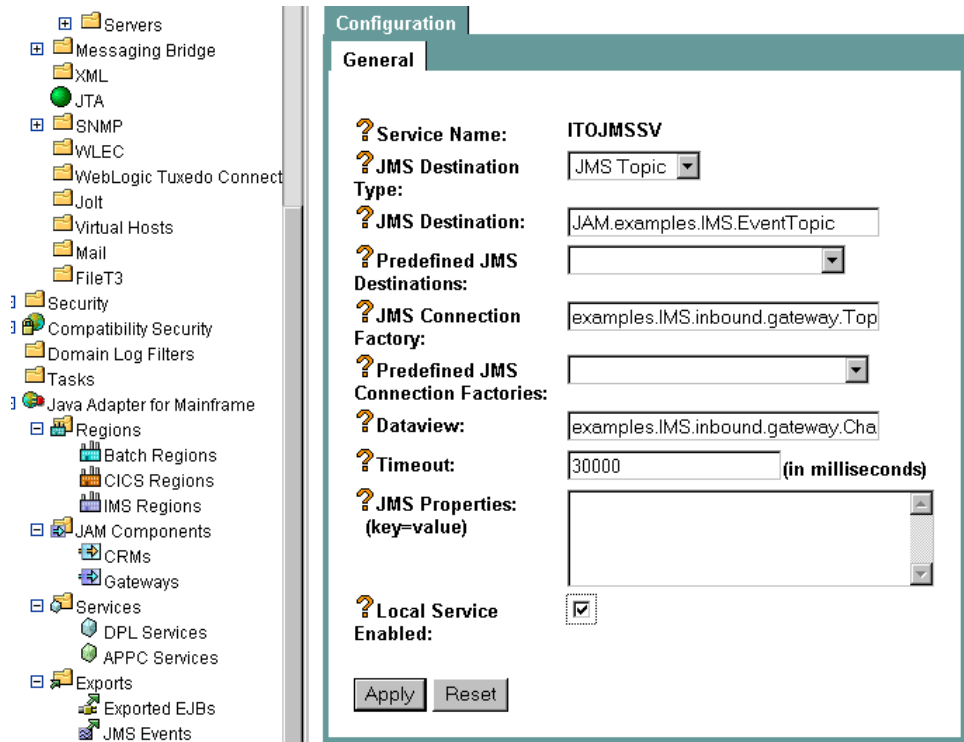
The files for the IMS side of the sample are as follows:

File Name	File Purpose
COMPIMSC	JCL that compiles and links the IMSTOJMS program.
IMSINDEF	Contains sample IMS stage 1 input and PSBGEN for the IMS configuration of the IMSTOJMS program.
DFS62DTI	IMS LU 6.2 descriptor that maps LTERM name to Logical Unit and remote transaction name.
IMSTOJMS	IMS client program that receives LTERM name and string from the user and queues request data on IMS message queue.

Procedure: How to Configure the Sample

To set up the IMS application to WebLogic Server JMS topic, complete the following steps.

1. Enable the JMS event:



- a. Click *Java Adapter for Mainframe*, *Exports*, and then *JMS Events* in the left pane.
 - b. Click *ITOJMSSV*.
 - c. Check *Local Service Enabled* and click *Apply* to enable the Local Service.
2. Set the environment by executing the `setExamplesEnv` command.

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

```
./setExamplesEnv.sh
```

The following message will display:

```
"Your environment has been set."
```

3. You may recompile the source to run your samples as follows:

Caution: Using the following options will overwrite files that are installed with the iWay JAM samples.

- a. Run the build.cmd (.sh) script to build the TopicReceive.class and the Chardata.class.
- b. Run egencobol to use the eGen Application Code Generator on chardata.egen to generate Chardata.java.

This option will generate the source. To compile the source, use the previous option to run the build.cmd (sh) script.

4. On the machine where the Gateway is installed, run the TopicReceiver program that listens on the JMS topic for incoming messages from IMS. At the command prompt, type:

```
java examples.IMS.inbound.gateway.TopicReceive "t3://hostname:port"
```

where:

hostname

Is the address of the machine where WebLogic Server is running.

port

Is the port for WebLogic Server.

For example, if TopicReceive is run on the same machine as WebLogic Server examples domain, the statement is:

```
java examples.IMS.inbound.gateway.TopicReceive "t3://localhost:7001"
```

5. Complete Mainframe tasks. On the machine with the IMS region:
 - a. Create a Partitioned Data Set (PDS) to store the source and JCL for this sample.
 - b. From the machine where the Gateway was installed, use FTP to send the following files to the PDS that you created:

IMSTOJMS

IMSINDEF

DFS62DTI

COMPIMSC

- c. In the COMPIMSC JCL, make the following changes:
 - a. Change the JOB statement.
 - b. Change YOURHLQ.SOURCE to the PDS you created.
 - c. Change YOUR.PGMLIB to a PGMLIB for your IMS region.
 - d. Change YOUR.PROCLIB to the location of CBLTDLI (usually IMS.PROCLIB).
 - e. Change YOUR.RESLIB to the proper value.

- f. Change RESLIB to SDFSRESL if you are using a newer IMS version. Refer to CBLTDLI in your PGMLIB to determine which library name your version uses. Make sure that if your IMS version requires this change, that you make the change to both the DD name and the library name.
6. Submit the COMPIMSC JCL. Ensure that all condition codes are 0.
7. Define the program IMSTOJMS to the IMS region. IMSINDEF contains sample IMS stage 1 and PSBGEN input. See your IMS systems programmer for assistance.
8. Define the LTERM. DFS62DTI contains sample definition for an APPC LTERM. See your IMS systems programmer for assistance.
 - a. Change the Logical Unit name to match the Logical Unit configured for the CRM in *Preparing to Use the IMS Samples* on page 2-4.
 - b. If you change the LTERM name, ensure you change it at the command prompt when you run the sample. See *How to Run the Sample* on page 2-16.
 - c. Do not change the TPNAME. It is configured to match the name of the JMSEVENT in the iWay JAM configuration.

Procedure: How to Run the Sample

To run the sample, complete the following steps:

1. Log on to the IMS region.
2. Type the following command in the IMS terminal:

```
IMSTOJMS JAMIMS01 hello
```

You can also use any other string. The string appear in the XML that is printed in the shell where the TopicReceive is running.

Note: JAMIMS01 is the LTERM name in the LU6.2 descriptor DFS62DTI. If you changed the LTERM name in *How to Configure the Sample* on page 2-13, you must enter that LTERM name at the command prompt instead of JAMIMS01.

3. To shut down the TopicReceive, type the following command in the IMS terminal:

```
IMSTOJMS JAMIMS01 quit
```

Using the Java Client to IMS Sample Application

After completing the steps in *Preparing to Use the IMS Samples* on page 2-4, you are ready to configure and run the Java client to IMS sample application.

Understanding How the Sample Works

This sample demonstrates requests from a Java client through iWay JAM to a remote service provided by an IMS application. The back-end application for this sample is the installation verification transaction, IVTNO, that is shipped with IMS. IVTNO is a non-conversational installation verification transaction that uses an OSAM database. This sample illustrates how Java applications may integrate with legacy IMS applications without modification to the IMS programs or configuration.

Understanding the Sample Configuration

No special configuration is required for this simple sample. The Java client calls the service, doIVTNO. The doIVTNO is an APPC service that is mapped to the IMS transaction, IVTNO. IVTNO is defined to IMS in the usual way. No special considerations are required for use with a Java client making requests through iWay JAM.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Four classes compose the WebLogic side of this sample application:

- IvtnoInRecord
- IvtnoOutRecord
- BaseClient
- Client

IvtnoInRecord and IvtnoOutRecord are DataView classes that are generated by the eGen Application Code Generator. The data members in these classes correspond to the data fields in the ivtno-in.cpy and ivtno-out.cpy copybooks. These copybooks match the input and output record layouts for the IVTNO transaction. These layouts are found in the IMS program DFSIVA1. IvtnoInRecord and IvtnoOutRecord are responsible for all data translation between the mainframe format of the data and the Java format of the data.

The BaseClient class that is generated by the eGen Application Code Generator is an extension of the EgenClient class. The callIVTNO method of BaseClient is a wrapper for calls to the callService method of the EgenClient class with doIVTNO as the service parameter in the call.

The Client class is the actual user interface. The Client class has a BaseClient member. The Client class receives a command and employee last name as command line parameters. The command must be one of the following: add, display, update, or delete. You may also enter an address and a port number if the iWay JAM Gateway is running on a different machine or the corresponding instance of WebLogic Server is listening on a different port than 7001. The URL is set in the BaseClient member. In the Client class, an IvtnoInRecord DataView is initialized with the input data. Depending on the command that you input, either the doDisplayOrDelete, doAdd, or doUpdate method is called. These methods, defined in the Client class, are wrappers for the callIVTNO method of BaseClient. The difference in the methods occur in the value set for the command as well as other fields of the input IvtnoInRecord. The returned IvtnoOutRecord DataView displays.

IMS Program

No IMS programs or configuration files are shipped with this sample because the sample uses the installation verification transaction, IVTNO, that is shipped with IMS.

Reference: Sample Files

The following table lists the sample files in this directory and their purpose:

File Name	File Purpose
ivtno-in.cpy	COBOL copybook that defines the structure of the input data for the IVTNO transaction.
ivtno-out.cpy	COBOL copybook that defines the structure of the output data from the IVTNO transaction.
ivtno.egen	eGen script that generates the IvtnoInRecord.java and IvtnoOutRecord.java DataView classes.
IvtnoInRecord.java	DataView class that corresponds to the ivtno-in.cpy COBOL copybook.
IvtnoOutRecord.java	DataView class that corresponds to the ivtno-out.cpy COBOL copybook.
baseClient.egen	eGen script that generates the IvtnoInRecord.java and IvtnoOutRecord.java DataView classes and the BaseClient.java EgenClient class.
BaseClient.java	Java class that extends EgenClient class that calls the IVTNO service.

File Name	File Purpose
Client.java	The user interface client class that receives a command and record name from the user, prompts the user for additional information, if necessary, and displays the result of the IVTNO service to the user. It invokes the IVTNO service by calling the callService method of its BaseClient member.

Procedure: How to Configure the Sample

To set up the Java client to IMS sample application, complete the following steps.

Note: This sample requires that IVTNO is installed and working in the IMS region before this sample is run. IVTNO should have been installed with the IBM IMS distribution. For information about IVTNO, see your IMS documentation.

1. Enable the service:

The screenshot shows a 'Configuration' dialog box with two tabs: 'General' and 'CRM Links'. The 'General' tab is active. It contains several fields with question mark icons to the left of the labels:

- Service Name:** doIVTNO
- Transaction Program:** IVTNO
- Id:** (empty field)
- Timeout:** 30000 (in milliseconds)
- Input Schema:** (empty field)
- Output Schema:** (empty field)
- Enabled:** ☒

At the bottom of the dialog are two buttons: 'Apply' and 'Reset'.

- a. In the left pane, click *Java Adapter for Mainframe, Services*, and then *APPC Services*.
 - b. In the right pane, click *doIVTNO* under *Service Name*.
 - c. Check *Enabled* and click *Apply*.
2. On the machine from which the sample client is run, set the environment.

Note: This machine does not have to be the machine on which the Gateway is running, but iWay JAM must be installed.

From a command prompt, switch to your examples domain and execute the following command to set the environment:

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

```
./setExamplesEnv.sh
```

The following message appears:

"Your environment has been set."

3. You may recompile the source to run your samples as follows:

Caution: Using the following options will overwrite files that are installed with the iWay JAM samples.

- a. Run the build.cmd (.sh) script to build the client classes.
- b. Run egencobol to use the eGen Application Code Generator on:
 - ivtno.egen to generate IvtnoInRecord.java and IvtnoOutRecord.java.
 - baseClient.egen to generate BaseClient.java, IvtnoInRecord.java, and IvtnoOutRecord.java.

This option will generate the source. To compile the source, use the previous option to run the build.cmd (sh) script.

For information about running the eGen Application Code Generator, see the iWay Java Adapter for Mainframe *Programming Guide*.

Running the Sample

To run the sample, type the following command at the command prompt:

```
java examples.IMS.outbound.gateway.Client [-m hostname] [-p port] -c  
command -n name
```

In this command, the following definitions apply:

- The first pair of command line switches are optional, and represent the `hostname` and `port` for a remote machine on which the iWay JAM Gateway is running in WebLogic Server. If you are running the client on the same machine as the Gateway and the WebLogic Server port is `7001`, then these options are not required.
- The second pair of command line switches are mandatory. The command must be one of the following:

- display
- add
- delete
- update
- name is the last name of the person that is the key to the record.

The following command is an example of a command that you might enter:

```
java examples.IMS.outbound.gateway.Client -c display -n LAST1
```

Using the Transactional Sample from WebLogic Server to IMS

After completing the steps in *Preparing to Use the IMS Samples* on page 2-4, you are ready to configure and run the transactional sample from WebLogic Server to IMS.

Understanding How the Sample Works

This sample demonstrates transactional requests made to an IMS application from a Java client through iWay JAM. This sample highlights client-initiated transactions that are distributed between an IMS-managed resource and a WebLogic Server-managed resource, a JMS queue. The back-end application for this sample is the IMS transaction, IVTNO. IVTNO is an IMS non-conversational installation verification transaction that uses an OSAM database. This sample illustrates how Java applications may be integrated with legacy IMS applications without modification to the IMS programs or configuration.

Understanding the Sample Configuration

No special configuration is required for this simple sample. The Java client calls the service doIVTNO. doIVTNO is an APPC service that is mapped to the IMS transaction, IVTNO. IVTNO is defined to IMS in the usual way and requires no special considerations to be used with a Java client making requests through iWay JAM.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Four classes compose the WebLogic side of this sample application:

- IvtnoInRecord
- IvtnoOutRecord
- BaseClient
- Client

IvtnoInRecord and IvtnoOutRecord are DataView classes that are generated by the eGen Application Code Generator. The data members in these classes correspond to the data fields in the ivtno-in.cpy and ivtno-out.cpy copybooks. These copybooks match the input and output record layouts for the IVTNO transaction. These layouts are in the IMS program, DFSIVA1. IvtnoInRecord and IvtnoOutRecord are responsible for all data translation between the mainframe format of the data and the Java format of the data.

The BaseClient class that is generated by the eGen Application Code Generator is an extension of the EgenClient class. The callIVTNO method of BaseClient is a wrapper for calls to the callService method of the EgenClient class with doIVTNO as the service parameter in the call.

The Client class is the actual user interface. The Client class has a BaseClient member. It has three optional command line parameters. You may enter an address and a port number if the iWay JAM Gateway is running on a different machine or the corresponding instance of WebLogic Server is listening on a different port than 7001. The URL is set in the BaseClient member. You may also enter a command line option that indicates that the distributed transaction in this sample should roll back. If this command line option is not used, the distributed transaction is committed.

doDisplayOrDelete, doAdd, and doUpdate methods are defined in the Client class. These methods are wrappers for the callIVTNO method of BaseClient. The difference in the methods occurs in the value set for the command as well as other fields of the input IvtnoInRecord.

The Client class first performs a check to make sure that the sample starts in a consistent state. The record that will be added to the OSAM database later is deleted by making a call to the doDisplayOrDelete method. The Client clears the JMS queue and then initiates the distributed transaction. The Client adds the record to the OSAM database using IVTNO by calling the doAdd method. The Client queues the record key on the JMS queue. Depending on your input, the Client then commits or rolls back the transaction. The Client class verifies the result by attempting to display the record from the OSAM database and the key from the JMS queue. Calling the doDisplayOrDelete method does the reading of the record from the OSAM record through IVTNO.

IMS Program

No IMS programs or configuration files are shipped with this sample because the sample uses the installation verification transaction, IVTNO, that is shipped with IMS.

Reference: Sample Files

The following table lists the sample files in this directory and their purpose:

File Name	File Purpose
ivtno-in.cpy	COBOL copybook that defines the structure of the input data for the IVTNO transaction.
ivtno-out.cpy	COBOL copybook that defines the structure of the output data from the IVTNO transaction.
ivtno.egen	eGen script that generates the IvtnoInRecord.java and IvtnoOutRecord.java DataView classes.
IvtnoInRecord.java	DataView class that corresponds to the ivtno-in.cpy COBOL copybook.
IvtnoOutRecord.java	DataView class that corresponds to the ivtno-out.cpy COBOL copybook.
baseClient.egen	eGen script that generates the IvtnoInRecord.java and IvtnoOutRecord.java DataView classes and the BaseClient.java EgenClient class.
BaseClient.java	Java class that extends EgenClient class that calls the IVTNO service.
Client.java	The user interface client class that receives a command line option to roll back or commit from the user. All invocations of the IVTNO transaction that it makes are made by calling the callService method of its BaseClient member. First, it deletes a record and clears the JMS queue. Then, it initiates a transaction. Within the boundaries of that transaction, it adds the record and queues the key to the record on the JMS queue. It then commits or rolls back the previous operations based on the command line option. It then verifies the operation by attempting to read the record and check the contents of the JMS queue.

Procedure: How to Configure the Sample

To set up the transactional sample from WebLogic Server to IMS, complete the following steps.

Note: The IMS installation verification sample (shipped with IMS), IVTNO, must be installed and working in the IMS region before running the sample.

1. Enable the service.

The screenshot shows a 'Configuration' window with two tabs: 'General' and 'CRM Links'. The 'General' tab is active. It contains the following fields and controls:

- Service Name:** doIVTNO
- Transaction Program Id:** IVTNO
- Timeout:** 30000 (in milliseconds)
- Input Schema:** (empty text box)
- Output Schema:** (empty text box)
- Enabled:** ☒
- Buttons:** 'Apply' and 'Reset'

- a. In the left pane, click *Java Adapter for Mainframe, Services*, and then *APPC Services*.
 - b. In the right pane, click *doIVTNO*.
 - c. Check *Enabled* and click *Apply*.
2. On the machine from which the sample client is to be run, set the environment.

Note: This machine does not have to be the machine on which the Gateway is running, but iWay JAM must be installed.

From a command prompt, switch to your examples domain and execute the following command to set the environment:

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

```
./setExamplesEnv.sh
```

The following message will display:

```
"Your environment has been set."
```

3. You may recompile the source to run your samples as follows:

Caution: Using the following options will overwrite files that are installed with the iWay JAM samples.

- a. Run the build.cmd (.sh) script to build the client classes.
- b. Run egencobol to use the eGen Application Code Generator on:
 - ivtno.egen to generate IvtnoInRecord.java and IvtnoOutRecord.java
 - baseClient.egen to generate BaseClient.java, IvtnoInRecord.java, and IvtnoOutRecord.java

This option will generate the source. To compile the source, use the previous option to run the `build.cmd` (.sh) script.

Running the Sample

To run the sample, type the following command at the command prompt:

```
java examples.transactional.IMS.outbound.gateway.Client
[-m hostname] [-p port] [-r]
```

In this command, the following definitions apply:

- The first pair of command line switches are optional, and represent the `hostname` and `port` for a remote machine on which the iWay JAM Gateway is running in WebLogic Server. If you are running the client on the same machine as the Gateway and the WebLogic Server port is 7001, then these options are not required.
- The third command line switch, which is also optional, indicates whether a rollback will be done or not. If `-r` is present at the command prompt, then the creation of the record on the database will be rolled back.

The following command is an example of a command that you might enter:

```
java examples.transactional.IMS.outbound.gateway.Client -r
```

Using the Transactional IMS to WebLogic Server

The sample contains the following files which can be used to make a transactional IMS application, which calls TOUPPER.

Sample File	Description
APPCCRM	The RRS/APPCC interface program.
APPCMVS	The APPCMVS definition for protected conversation LU.
CLNTEXEC	The Client execute JCL.
CCOMP	The APPCCRM compile JCL.
COBCOMP	The IMSBTCH compile JCL.
COBIMS	The JAMCPIC1 compile JCL.
IMSBTCH	The batch driver program to start JAMCPIC1 in IMS.
JAMCPIC1	The IMS CPI-C Driven program.
TPPROF	The CPI-C driven program Tp-profile definition.
VTAMDEF	The VTAM definition for protected conversation LU.

CHAPTER 3

Using the CICS Samples

Topics:

- About the CICS Samples
- Roadmap for the Samples
- Using the Samples

The CICS samples demonstrate how BEA WebLogic Java Adapter for Mainframe (iWay JAM) integrates the WebLogic applications with CICS applications.

About the CICS Samples

The following section provides a brief overview of each of the CICS samples described in this guide. A detailed description of how each sample works and instructions for running each sample are provided in the [“Using the Samples”](#) section.

CICS Application to WebLogic Server Sample EJB

This sample demonstrates the functional capability provided by iWay JAM to invoke the services of an Enterprise Java Bean (EJB) deployed with WebLogic Server from a CICS client. The server EJB is similar to the Trader bean that is shipped with WebLogic Server. The COBOL CICS client program makes a series of requests to the EJB to buy shares of stock.

As in the WebLogic Server sample, the EJB will check the number of shares requested against a preconfigured trading limit to decide if the requested number of shares can be purchased. If the number of shares is too high, then it will actually buy the limit instead. A record of each sale is entered into the WebLogic Server log. The bean will return to the client the actual number of shares purchased as well as the stock symbol. The CICS client will report the result to the screen.

Java Client to CICS Sample Application

This sample illustrates the invoking of CICS services from requests that originate from a Java client. Four COBOL CICS programs are provided. These programs create, read, update, and delete records from a simulated database. These programs simulate a database through the use of Temporary Storage (TS) queues. Employee records are stored in the database by creating a TS queue with a name that is the first eight characters of the employee's name. The use of TS queues to simulate a database requires no configuration in the CICS region other than the definition of the programs.

The Java client receives a command and a record name from you. The command is one of the following: `add`, `display`, `update`, or `delete`. You may choose to enter a host address and port if the gateway is running on another machine. Depending on the command, the client may prompt you for additional information. The client then makes a service call to one of the provided CICS COBOL programs. The result displays.

Transactional Sample from WebLogic Server to CICS

This sample illustrates making calls to remote services located in a CICS region from a Java client. The service calls that alter data on the mainframe occur within the boundaries of two-phase commit transactions. The sample contains a transaction that is distributed over resources managed by WebLogic Server and resources managed by CICS. This transaction uses a service call to add a record to a VSAM file. The key to the record is inserted on a JMS queue within the boundaries of the same transaction as a service call to create the record. The queuing of the record key and the creation of the record in VSAM will either be committed or rolled back together depending on the command line option you set.

Four COBOL CICS programs are provided. These programs create, read, update, and delete records from the VSAM file. Employee records are stored in the VSAM file using the employee's social security number as a key. The VSAM file is used as the recoverable resource in CICS because it is relatively easy to create and configure and CICS users will have VSAM.

Roadmap for the Samples

To run the CICS samples, follow the roadmap listed below. General tasks for all of the CICS samples include:

1. Verify prerequisite tasks.

For a listing of prerequisite tasks, see Chapter 1, *iWay Java Adapter for Mainframe Samples Overview*.

2. Prepare to use the CICS sample.

- a. Start the CRM.
- b. Update the iWay JAM configuration file.
- c. Start the `examples` domain.
- d. Configure the iWay JAM Gateway.

Specific tasks for each sample include:

1. Set up the sample.

- a. Enable services.
- b. Set the environment.
- c. Generate and build source (optional).
- d. Complete mainframe tasks.

2. Run the sample.

Using the Samples

After you have completed the tasks described in Chapter 1, *iWay Java Adapter for Mainframe Samples Overview* section, you are ready to use the sample.

Preparing to Use the CICS Samples

The following steps are common to all the CICS samples. These steps only need to be performed once for all CICS samples.

Step 1: Start the CRM

Before starting the iWay JAM Gateway, start the CRM. The CRM must be configured with certain parameter values at startup. These parameter values include:

- The address of the machine on which the CRM is running
- The port on which the CRM listens
- The name the Gateway will use to refer to the CRM

For running the samples, you must set the machine address and port. The values that you set for the machine address and port when the CRM is started, must agree with the values that you set for the CRM in the WebLogic Administration Console for the samples CRM. The name of the CRM that is preconfigured for running all of the samples is CRM1. Use this name when the CRM is started to run any of the samples.

The way you start the CRM depends on whether the CRM will be started under a Unix or MVS system. On Unix, start the CRM using a shell script. On MVS, start the CRM using JCL. For more information, see the iWay Java Adapter for Mainframe Configuration and Administration Guide.

Step 2: Update the iWay JAM Configuration File

On the machine where the Gateway is located, switch to your examples domain by copying `jamconfig_CICS.xml` to `jamconfig.xml`.

Step 3: Start the examples Domain

From the command prompt, execute the following command from the same directory to start the `examples` domain:

- 1 For Microsoft Windows:

```
startExamplesServer.cmd
```
- 1 For Unix:

```
./startExamplesServer.sh
```

Step 4: Configure the iWay JAM Gateway

Most configuration tasks are preconfigured or were completed during the installation process by the installer program. For additional information about configuring iWay JAM, see the iWay Java Adapter for Mainframe Configuration and Administration Guide. Make the following configuration changes for the CICS Sample to run on your system. These changes can be made in the WebLogic Administration Console in the following way.

1. From your browser, open the WebLogic Administration Console using the following address:

`http://hostname:7001/console`

In this address, the following definitions apply:

`hostname` is the address of the machine where WebLogic Server is running.

`7001` is the port for WebLogic Server that has been configured for the `examples` domain.

2. When prompted, supply the following user and password information:

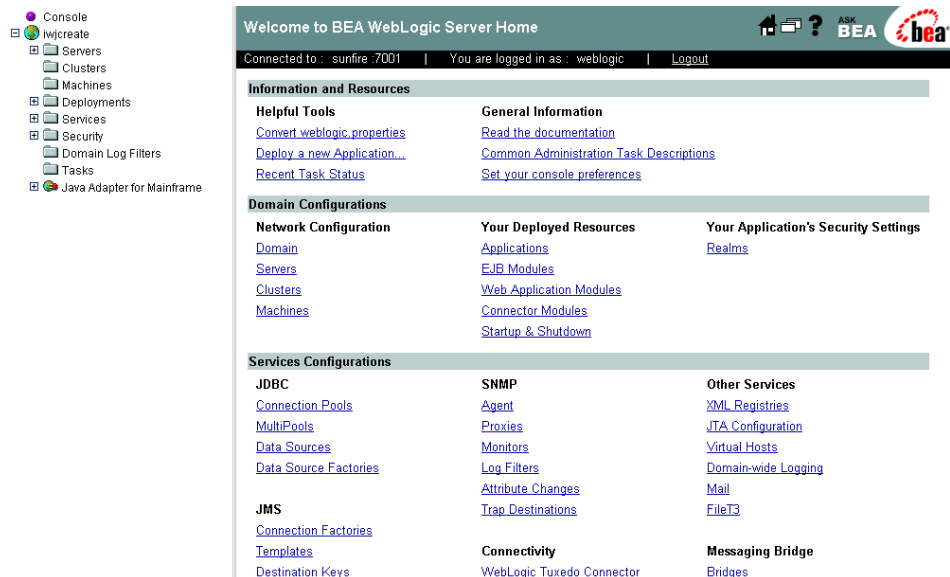
1 user: `system`

This user name cannot be changed.

1 password: `security`

To change the password, see the BEA WebLogic Server documentation.

The WebLogic Administration Console displays.



3. To configure the CRM to the iWay JAM Gateway, complete the following steps:
 - a. In the left pane, click on **Java Adapter for Mainframe** → **JAM Components** → **CRMs**. In the right pane, click **CRM1**. On the **General** tab, set the following fields to correspond with your system. Click **Apply**. When the CRM is active, **Status** turns from red to green.

ConfigurationAdministration

GeneralLinks

? **Name:** CRM1

? **Status:** ● Up

? **Listen Address:**

? **Listen Port:**

? **Logical Unit:**

? **Stack Type:**

Note: If you specify a different logical unit for this CRM, you will be unable to recover any pending transactions between WebLogic Server and the regions that were linked to CRM1.

Field	Field Description
Listen Address	The address of the machine where the CRM is installed and running. This address must match the address set in the CRM startup JCL or script.
Listen Port	The port for the CRM. This entry must match the port set in the CRM startup JCL or script.
Logical Unit	The name of the Logical Unit defined for the CRM.
Stack Type	The stack type.

- b. To configure the CICS Region, click **Java Adapter for Mainframe** → **Regions** → **CICS Regions** in the left pane. In the right pane, click on **CICS3** and enter the name of the **Logical Unit**. Click **Apply** to set the Logical Unit.

Note: This Logical Unit is the `ACBNAME` in the VTAM Logical Unit definition or the VTAM `APPLID` of the region. This Logical Unit is not the same as the Logical Unit for the CRM in (3a).

The screenshot shows a configuration window titled "Configuration" with a "General" tab selected. Inside the tab, there are two fields: "Name:" with the value "CICS3" and "Logical Unit:" with the value "CICSLU". Below these fields are two buttons: "Apply" and "Reset". At the bottom of the dialog, there is a note: "Note: If you specify a different logical unit for region CICS3, you will be unable to recover any transactions that were pending between WebLogic Server and this region."

Configuration

General

? **Name:** CICS3

? **Logical Unit:**

Note: If you specify a different logical unit for region CICS3, you will be unable to recover any transactions that were pending between WebLogic Server and this region.

- c. Click **CICS Regions** at the top of the pane. Click **CRM1CICS3**. In the new window, click **CRM1CICS3**. On the **Links** tab, check **Deployed** and click **Apply**.

The screenshot shows a configuration window for CRM1CICS3. At the top, there are two tabs: 'Configuration' and 'Administration'. Below these, there are two sub-tabs: 'General' and 'Links'. The 'Links' tab is currently selected. The main area of the window contains a list of configuration parameters for the link CRM1CICS3. Each parameter is preceded by a question mark icon. The parameters and their values are: Name: CRM1CICS3; Status: Down (indicated by a red circle icon); Region Name: CICS3 (CICSRegion) (selected from a dropdown); Logmode Name: SMSNA100; Maximum Sessions: 4; Minimum Winner Sessions: 2; Security Level: Local (selected from a dropdown); Default User-Id: (empty text field); Deployed: checked (checkbox). At the bottom of the configuration area, there are two buttons: 'Apply' and 'Reset'. Below the buttons, there is a note: 'Note: By modifying the parameters of this link you will be unable to recover any transactions that were pending on this link between WebLogic Server and region CICS3.'

? Name:	CRM1CICS3
? Status:	Down
? Region Name:	CICS3 (CICSRegion) ▼
? Logmode Name:	SMSNA100
? Maximum Sessions:	4
? Minimum Winner Sessions:	2
? Security Level:	Local ▼
? Default User-Id:	
? Deployed:	☒

Note: By modifying the parameters of this link you will be unable to recover any transactions that were pending on this link between WebLogic Server and region CICS3.

- d. In the left pane, click **JAM Components**→**Gateways**. Click **JAM5.1** in the right pane. On the **General** tab, check **Deployed**. Click **Apply**.

The screenshot shows the 'Configuration' window with the 'Administration' tab selected. The 'General' sub-tab is active. The configuration for gateway 'JAM5.1' is displayed. The 'Status' is 'Down' (indicated by a red circle). The 'Target WebLogic Server' is 'exampleserver' and the 'Target CRM' is 'CRM1'. The 'Deployed' checkbox is checked. There are 'Apply' and 'Reset' buttons at the bottom. A note at the bottom states: 'Note: By modifying the Target CRM parameter of this gateway you will be unable to recover any transactions that were pending for gateway JAM5.1.'

? Name:	JAM5.1
? Status:	● Down
? Target WebLogic Server:	exampleserver ▼
? Target CRM:	CRM1 ▼
? Deployed:	☒

Apply Reset

Note: By modifying the Target CRM parameter of this gateway you will be unable to recover any transactions that were pending for gateway JAM5.1.

4. To start the iWay JAM Gateway, select the **Administration** tab → **Start/Stop** tab. Click **Start** to start the Gateway.

The screenshot shows the 'Administration' window with the 'Start / Stop' sub-tab selected. The 'Start / Stop Gateway' section displays the gateway name 'JAM5.1' and its status 'Down' (indicated by a red circle). There are 'Start' and 'Stop' buttons at the bottom.

? Name:	JAM5.1
? Status:	● Down

Start Stop

If the Gateway is running, **Status** changes to green in the WebLogic Administration Console and the following message is recorded in the WebLogic Server log:

“JAM Gateway ready for use. Current link status: up(1).”

You have completed the general steps required to prepare your system to run the CICS samples. Select the CICS sample you want to run and follow the steps in that section to set up and run that sample.

Using the CICS Application to WebLogic Server Sample EJB

After completing the steps in *Preparing to Use the CICS Samples* on page 3-3 section, you are ready to set up and run the CICS application to WebLogic Server sample EJB.

Understanding How the Sample Works

This sample demonstrates the functional capability of iWay JAM to invoke the services of an EJB deployed in WebLogic Server from a CICS client. The server EJB is similar to the stateless session Trader bean that is shipped with WebLogic Server as an example. The COBOL CICS client program makes a series of requests to the EJB to buy shares of stock.

Understanding the Sample Configuration

The CICS COBOL client program `TRADCLNT` is defined to CICS in the standard way any program is defined to CICS. No changes are necessary to use this program as a client making requests through iWay JAM. `TRADCLNT` does a Distributed Program Link (DPL) to the remote service `TRADSERV`. `TRADSERV` is defined to the CICS region in the standard manner for defining remote services to a CICS region; however, the `REMOTESYSTEM` parameter in the definition must be set to the name of the connection defined to the CICS region for the CRM. The `REMOTENAME` in the definition of `TRADSERV` is also set to `TRADSERV`.

In the iWay JAM configuration, `TRADSERV` is the name of an `EJBExport` that is mapped to the JNDI name `jam.TradeServer`. `jam.TradeServer` is the value of the JNDI-name element in the WebLogic deployment descriptor `weblogic-ejb-jar.xml` for the `TradeServer` EJB.

The eGen Application Code Generator generates deployment descriptors when it generates the code for EJBs. However, to avoid name collisions that can occur when multiple EJBs are generated from a single eGen script, generated deployment descriptors are always given names that contain the stem name of the EJB that is being generated. In this sample, the generated deployment descriptors are named `TradeServer-jar.xml` and `wl-TradeServer-jar.xml`. Before these deployment descriptors can be used for an actual deployment in WebLogic Server, they have to be renamed `ejb-jar.xml` and `weblogic-ejb-jar.xml`. Because the generated EJB `TradeServerBean` is extended in this sample, the value of the `ejb-class` element in `ejb-jar.xml` must also be manually changed to the name of the extension class that contains the business logic, `ExtTradeServerBean`.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Five classes compose the WebLogic side of this sample application:

- `TradeRecord`
- `TradeServer`
- `TradeServerBean`
- `TradeServerHome`
- `ExtTradeServerBean`

`TradeRecord` is a `DataView` class, generated by the eGen Application Code Generator. The data members in the `TradeRecord` class correspond to the data fields in the `TRADRCRD` copybook. The `TradeRecord` class is responsible for all data translation between the mainframe format of the data and the Java format of the data.

`TradeServer`, `TradeServerBean`, and `TradeServerHome` classes are generated by the eGen Application Code Generator. `TradeServer` is a remote interface that contains the definition of a single method: `dispatch`. `dispatch` is the essential method for server EJBs that are used in iWay JAM applications. This method is messaged by the Gateway when a mainframe client makes a request of the corresponding service. `TradeServerHome` is a standard home interface for a stateless session bean. It contains the definition of a `create` method that returns a `TradeServer` object to the caller. `TradeServerBean` extends `EgenServerBean`. `TradeServerBean` contains the implementation of the `dispatch` method that is a wrapper for the `buy` method. The implementation of the `buy` method that is given in `TradeServerBean` does not perform actual business logic. As it is defined in the eGen script that generates `TradeServerBean`, the `buy` method only receives a `TradeRecord` object and returns a `TradeRecord` object. To actually do any business logic, the `TradeServerBean` must be extended and the `buy` method overwritten.

The extension of the `TradeServerBean` class that is included with this sample is called `ExtTradeServerBean`. `ExtTradeServerBean` contains an implementation of the `buy` method containing the business logic. The number of shares that are requested is compared to a predefined limit. If the number of shares is greater than this limit, then the number of shares is reset to the limit. The purchase of the shares is recorded in the WebLogic Server log.

CICS Program

The program `TRADCLNT` is a simple COBOL CICS client program that creates several `TRADRCRD` records and does a DPL to the remote service `TRADSERV` for each record. Each one of the records represents a request to purchase some stock. No special considerations are required in this program as a result of linking through iWay JAM to a service offered by an EJB deployed in WebLogic Server.

Reference: Sample Files

The following table lists the sample files and their purpose:

File Name	File Purpose
TradeRecord.cpy	COBOL copybook that defines the structure of the mainframe data.
tradeserver.egen	eGen script that generates the TradeRecord.java DataView class, the TradeServer.java, TradeServerBean.java, TradeServerHome.java bean classes, and the TradeServer-jar.xml and wl-TradeServer-jar.xml deployment descriptors.
TradeRecord.java	DataView class that corresponds to the TradeRecord.cpy COBOL copybook.
TradeServer.java	EJB remote interface generated by the eGen utility.
TradeServerBean.java	EJB generated by the eGen utility.
TradeServerHome.java	EJB home interface generated by the eGen utility.
ExtTradeServerBean.java	EJB that extends TradeServerBean. The business logic of servicing the requests from the mainframe is implemented in this class.
TradeServer-jar.xml	Deployment descriptor generated by tradeserver.egen.
wl-TradeServer-jar.xml	WebLogic deployment descriptor generated by tradeserver.egen.
ejb-jar.xml	TradeServer-jar.xml deployment descriptor that has been modified and renamed for inclusion in the JAM_TradeServer.jar.
weblogic-ejb-jar.xml	wl-TradeServer-jar.xml WebLogic deployment descriptor renamed for inclusion in the JAM_TradeServer.jar.

The following table lists the sample files and their purpose:

File Name	File Purpose
TRADRCRD	COBOL copybook that defines the structure of the mainframe data.
CMPPROC	Procedure used to compile and link the CICS programs.
COMPTRCL	JCL that executes the CMPPROC for the program TRADCLNT.
CSDUPDTR	RDO cards to define TRADCLNT and the service TRADSERV to the CICS region.
CSDUTRCL	JCL that executes CSDUPDTR.
TRADCLNT	CICS client program that makes requests to purchase several stocks to the TRADSERV service. This service is mapped to the buy method of the ExtTradeServerBean by iWay JAM.

Setting Up the Sample

To set up the CICS application to WebLogic Server sample EJB, complete the following steps.

Step 1: Enable the Service

To enable the Exported EJB, click **Java Adapter for Mainframe** → **Exports** → **ExportedEJBs** in the left pane. In the right pane, click **TRADSERV**. Check **Local Service Enabled**. Click **Apply** to enable the Local Service.

The image shows a 'Configuration' dialog box with a 'General' tab. It contains the following fields and controls:

- Service Name:** TRADSERV
- JNDI Name:** jam.TradeServer
- Timeout:** 30000 (in milliseconds)
- Local Service Enabled:** ☒
- Buttons:** Apply, Reset

Step 2: Set the Environment

On the machine where the Gateway is installed, set the environment by completing the following step:

- n From a command prompt, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples` and execute the following command to set the environment:

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

```
./setExamplesEnv.sh
```

The following message will display:

“Your environment has been set.”

Step 3: Complete Mainframe Tasks

On the machine with the CICS region:

1. Create a Partitioned Data Set (PDS) to store the source and JCL for this sample.
2. From the machine where the Gateway was installed, FTP the following files into the PDS that you created:
 - 1 CMPPROC
 - 1 COMPTRCL
 - 1 CSDUPDTR
 - 1 CSDUTRCL
 - 1 TRADCLNT
 - 1 TRADRCRD
3. In the procedure CMPPROC, do not set LNKLIB, PDSSRC, and PROG. These settings are supplied by the COMPTRCL JCL that will exec COMPROC. Do set INDEX, COMPHLQ, COMPHL2. You may need to change OUTC and the unit of WORK.
4. In the COMPTRCL JCL, make the following changes:
 - 1 Change the JOB statement.
 - 1 Change the JCLLIB ORDER and the PDSSRC to the PDS you created.
 - 1 Change the LNKLIB to a LOADLIB for your CICS region.
5. Submit the COMPTRCL JCL. Make sure that the condition code is 0.
6. In the RDO script CSDUTRCL, make the following changes:
 - 1 Change the name of the LIST to a valid list name for your CICS region.
 - 1 Change the value of REMOTESYSTEM to the connection defined for the CRM.
 - 1 If you change the GROUP name, make sure you change it everywhere.
7. In the CSDUPDTR JCL, make the necessary changes to these statements:
 - 1 JOB
 - 1 STEPLIB
 - 1 DFHCSD
 - 1 SYSIN DSN
8. Submit the CSDUPDTR JCL.

You may get a warning on the DELETE step, because the program TRADCLNT probably was not defined before. The condition code should not be more than 4.
9. Log on to your CICS region.
10. Install the iWay JAM sample.

To install the iWay JAM sample, type the following command at the command prompt:

```
CEDA INSTALL GROUP (JAMEXMPL)
```

11. Verify the CICS sample program.

To verify the CICS sample program, type the following command at the command prompt:

```
CEMT INQUIRE PROG (TRADCLNT)
```

Running the Sample

To run the sample, type the following command at the command prompt:

```
TRCL
```

TRCL is the transaction that is defined to the CICS region to execute the TRADCLNT program. You will see the buys being processed in the WebLogic Server log. The CICS terminal will report success or failure.

Using the Java Client to CICS Sample Application

After completing the steps in *Preparing to Use the CICS Samples* on page 3-3, you are ready to set up and run the Java client to CICS sample application.

Understanding How the Sample Works

This sample demonstrates requests from a Java client through iWay JAM to a remote service provided by a CICS application.

Understanding the Sample Configuration

This simple sample requires no special configuration. The Java client calls one of the services: sampleCreate, sampleRead, sampleUpdate, or sampleDelete. These DPL services are mapped to the CICS programs: DPLDEMOC, DPLDEMOR, DPLDEMOU, and DPLDEMOD. These programs are defined to CICS in the standard way any program is defined to CICS. No special considerations are necessary for using this program with a Java client making requests through iWay JAM.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Three classes compose the WebLogic side of this sample application:

- EmployeeRecord
- BaseClient

- `Client`

`EmployeeRecord` is a `DataView` class that is generated by the eGen Application Code Generator. The data members in the `EmployeeRecord` class correspond to the data fields in the `EMPREC` copybook. The `EmployeeRecord` class is responsible for all data translation between the mainframe format of the data and the Java format of the data.

The `BaseClient` class that is generated by the eGen Application Code Generator is an extension of the `EgenClient` class. The `newEmployee`, `readEmployee`, `updateEmployee`, and `deleteEmployee` methods of `BaseClient` are wrappers for calls to the `callService` method of the `EgenClient` class with `sampleCreate`, `sampleRead`, `sampleUpdate`, or `sampleDelete`, as service parameters in the call.

The `Client` class is the actual user interface. The `Client` class has a `BaseClient` member. The `Client` class receives a command and employee last name as command line parameters. The command must be one of the following: `add`, `display`, `update`, or `delete`. You may also enter an address and a port number if the iWay JAM Gateway is running on a different machine or the corresponding instance of WebLogic Server is listening on a different port than 7001. The URL is set in the `BaseClient` member. In the `Client` class an `EmployeeRecord DataView` is initialized with the input data. Depending on the command that you input, the `doAdd`, `doDisplay`, `doUpdate`, or `doDelete` method is called. These methods that are defined in the `Client` class are wrappers for the `newEmployee`, `readEmployee`, `updateEmployee`, and `deleteEmployee` methods of `BaseClient`. The `EmployeeRecord DataView` that is returned as a result of the operation is displayed to you.

CICS Programs

Four CICS COBOL programs are included with this sample:

- `DPLDEMOC`
- `DPLDEMOR`
- `DPLDEMOU`
- `DPLDEMOD`

These programs imitate the four basic operations for database records: insert, read, update, and delete. Temporary Storage (TS) queues are used in this sample to simulate a database. For example, to imitate the operation of inserting a record in a table in a database, a TS queue is created with a key that is the first eight characters of the last name field in the data. To read the record, the TS queue is read. These programs are simple but ordinary CICS COBOL server programs that are linked to and passed a `COMMAREA`. The necessary TS queue operation is done using the data record in the `COMMAREA`. The structure of the data in the `COMMAREA` is given in the copybook `EMPREC`. No special considerations are required in this program as a result of being used in an application with a Java client making requests through iWay JAM.

Reference: Sample Files

The following table lists the sample files and their purpose:

File Name	File Purpose
<code>emprec.cpy</code>	COBOL copybook that defines the structure of the mainframe employee record.
<code>emprec.egen</code>	eGen script that generates the <code>EmployeeRecord.java</code> <code>DataView</code> class.
<code>EmployeeRecord.java</code>	<code>DataView</code> class that corresponds to the <code>emprec.cpy</code> COBOL copybook.
<code>baseClient.egen</code>	eGen script that generates the <code>EmployeeRecord.java</code> <code>DataView</code> class and the <code>BaseClient.java</code> <code>EgenClient</code> class.
<code>BaseClient.java</code>	Java class that extends <code>EgenClient</code> class that calls the various mainframe services.
<code>Client.java</code>	The user interface client class that receives a command and record name from the user, prompts the user for additional information if necessary, and displays the result of the mainframe service calls to the user. It invokes the mainframe services by calling the <code>callService</code> method of its <code>BaseClient</code> member.

The files for the CICS side of the sample are as follows:

File Name	File Purpose
<code>EMPREC</code>	COBOL copybook that defines the structure of the employee record data.
<code>DPLDEMOC</code>	CICS server program that imitates the insertion of a record on a database by creating a TS queue.
<code>DPLDEMOR</code>	CICS server program that imitates the reading of a record on a database by reading a TS queue.
<code>DPLDEMOU</code>	CICS server program that imitates the update of a record on a database by updating a TS queue.
<code>DPLDEMOD</code>	CICS server program that imitates the deletion of a record on a database by deleting a TS queue.

File Name	File Purpose
CMPPROC	Procedure used to compile and link the CICS programs.
COMPCRUND	JCL that executes the CMPPROC for the programs DPLDEMOC, DPLDEMOR, DPLDEMOU, and DPLDEMOD.
CSDUCRUND	RDO cards to define DPLDEMOC, DPLDEMOR, DPLDEMOU, and DPLDEMOD to the CICS region.
CSDUPDCO	JCL that executes CSDUCRUND.

Setting Up the Sample

To set up the Java client to CICS sample application, complete the following steps.

Step 1: Enable the Services

To enable the DPL Services, click **Java Adapter for Mainframe** → **Services** → **DPLService** in the left pane. Click **sampleCreate**. Check **Enabled** and click **Apply** to enable the Local Service.

Repeat this process for each of the following DPL services:

- sampleRead
- sampleUpdate
- sampleDelete

Configuration

General CRM Links

? **Service Name:** sampleCreate

? **CICS Program Name:** DPLDEMOC

? **Alternate Transaction Id:**

? **Timeout:** 30000 (in milliseconds)

? **Input Schema:**

? **Output Schema:**

? **Enabled:** ☒

Apply Reset

Step 2: Set the Environment

On the machine where the Gateway is installed, set the environment by completing the following step:

- n From a command prompt, execute the following command to set the environment:

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

```
./setExamplesEnv.sh
```

The following message displays:

“Your environment has been set.”

Step 3: Complete Mainframe Tasks

On the machine with the CICS region:

1. Create a Partitioned Data Set (PDS) to store the source and JCL for this sample.

2. From the machine where the Gateway was installed, FTP the following files into the PDS that you created:
 - EMPREC
 - DPLDEMOC
 - DPLDEMOR
 - DPLDEMOU
 - DPLDEMOD
 - CMPPROC
 - COMPCRUD
 - CSDUCRUD
 - CSDUPDCO
3. In the procedure CMPPROC, do not set LNKLIB, PDSSRC, and PROG. The values are supplied by the COMPCRUD JCL that will execute COMPROC. Do set INDEX, COMPHLQ, and COMPHL2. You may need to change OUTC and the unit of WORK.
4. In the COMPCRUD JCL, make the following changes:
 - 1 Change the JOB statement.
 - 1 Change the JCLLIB ORDER and the source to the PDS you created.
 - 1 Change the LOADLIB to a load lib for your CICS region.
5. Submit the COMPCRUD JCL. Make sure that the condition code is 0.
6. In the RDO script, CSDUCRUD, make the following changes:
 - 1 Change the name of the LIST to a valid list name for your CICS region.
 - 1 You can also change the GROUP name, but make sure you change it everywhere.
7. In the CSDUPDCO JCL, make the following changes:
 - 1 Change the JOB statement.
 - 1 Change the STEPLIB.
 - 1 Change the DFHCSD.
 - 1 Change the SYSIN DSN.
8. Submit the CSDUPDCO JCL.

You may get a warning on the DELETE step, because the programs probably were not defined before. The condition code should not be higher than 4.

9. Log on to your CICS region.

10. Install the iWay JAM sample.

To install the iWay JAM sample, type the following command at the command prompt:

```
CEDA INSTALL GROUP(JAMEXMPL)
```

11. Verify the CICS sample programs.

To verify the CICS sample programs, type the following command at the command prompt:

```
CEMT INQUIRE PROG(DPLDEMOC)
```

Repeat for DPLDEMOR, DPLDEMOU, and DPLDEMOD.

Running the Sample

To run the sample, type the following command at the command prompt:

```
java examples.CICS.outbound.gateway.Client [-m hostname] [-p port] -c  
'command' -n 'name'
```

In this command, the following definitions apply:

- n The first pair of command line switches are optional and represent the `hostname` and `port` for a remote machine on which the iWay JAM Gateway is running in WebLogic Server. If you are running the client on the same machine as the Gateway and the WebLogic Server port is 7001, these options are not required.
- n The second pair of command line switches are mandatory.
 - l `command` must be one of the following: display, add, delete, update.
 - l `name` is the last name of the person that is the key to the record.

The following command is an example of a command that you might enter:

```
java examples.CICS.outbound.gateway.Client -c add -n Wilson
```

Note: Your CICS administrator may want to delete any TS queues that remain after running this sample.

Using the Transactional Sample from WebLogic Server to CICS

After completing the steps in *Preparing to Use the CICS Samples* on page 3-3, you are ready to set up and run the transactional sample from WebLogic Server to CICS.

Understanding How the Sample Works

This sample demonstrates transactional requests made to a CICS application from a Java client through iWay JAM. This sample highlights client-initiated transactions that are distributed between a CICS-managed resource, a VSAM file, and a WebLogic Server-managed resource, a JMS queue.

Understanding the Sample Configuration

This simple sample requires no special configuration. The Java client calls one of the services: `sampleCreateT`, `sampleReadT`, `sampleUpdateT`, or `sampleDeleteT`. These DPL services are mapped to the CICS programs: `DPLDEMVC`, `DPLDEMVR`, `DPLDEMVB`, and `DPLDEMVD`. These programs are defined to CICS in the standard way any program is defined to CICS. The VSAM file used in this sample is created and defined in the standard manner. No special considerations need to be made for use with a Java client making requests through iWay JAM.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Three classes make up the WebLogic side of this sample application:

- `EmployeeRecord`
- `BaseClient`
- `Client`

`EmployeeRecord` is a `DataView` class that is generated by the eGen Application Code Generator. The data members in the `EmployeeRecord` class correspond to the data fields in the `EMPREC` copybook. The `EmployeeRecord` class is responsible for all data translation between the mainframe format of the data and the Java format of the data.

The `BaseClient` class that is generated by the eGen Application Code Generator is an extension of the `EgenClient` class. The `newEmployee`, `readEmployee`, `updateEmployee`, and `deleteEmployee` methods of `BaseClient` are wrappers for calls to the `callService` method of the `EgenClient` class with `sampleCreateT`, `sampleReadT`, `sampleUpdateT`, or `sampleDeleteT`, as service parameters in the call.

The `Client` class is the actual user interface. The `Client` class has a `BaseClient` member. It uses three optional command line parameters. You may enter an address and a port number if the iWay JAM Gateway is running on a different machine or the corresponding instance of WebLogic Server is listening on a different port than 7001. The URL is set in the `BaseClient` member. You may also enter a command line option that indicates that the distributed transaction in this sample should be rolled back. If this command line option is not used, the distributed transaction will be committed.

The `Client` class first performs a check to make sure that the sample starts in a consistent state. The record that will later be added to the VSAM file is deleted by making a call to the `deleteEmployee` method of the `BaseClient` member. The `Client` then clears the JMS queue and then initiates the distributed transaction. The `Client` adds the record to the VSAM file by calling the `newEmployee` method of the `BaseClient` member. The `Client` queues the record key on the JMS queue. Depending on your input, the `Client` commits or rolls back the transaction. The `Client` class

verifies the result by attempting to display the record from VSAM file and the key from the JMS queue. Calling the `readEmployee` method of the `BaseClient` member results in the reading of the record from the VSAM file.

CICS Programs

Four CICS COBOL programs are included with this sample:

- DPLDEMVC
- DPLDEMVR
- DPLDEMVCU
- DPLDEMVD

These programs imitate the four basic operations that can be done with records in a database: insert, read, update, and delete. A VSAM file is used in this sample in place of a database. A VSAM file is used as the recoverable resource in CICS because it is relatively easy to create and configure. For example, to imitate the operation of inserting a record in a table in a database a record is created in the VSAM file with a key that is the social security number field in the data. These programs are simple but ordinary CICS COBOL server programs that are linked to and passed a `COMMAREA`. The necessary VSAM file operation is done using the data record in the `COMMAREA`. The structure of the data in the `COMMAREA` is given in the copybook `EMPREC`. No special considerations are required in this program as a result of being used in an application with a Java client making requests through iWay JAM.

Reference: Sample Files

The following table lists the sample files and their purpose:

File Name	File Purpose
<code>emprec.cpy</code>	COBOL copybook that defines the structure of the mainframe employee record.
<code>emprec.egen</code>	eGen script that generates the <code>EmployeeRecord.java</code> <code>DataView</code> class.
<code>EmployeeRecord.java</code>	<code>DataView</code> class that corresponds to the <code>emprec.cpy</code> COBOL copybook.
<code>baseClient.egen</code>	eGen script that generates the <code>EmployeeRecord.java</code> <code>DataView</code> class and the <code>BaseClient.java</code> <code>EgenClient</code> class.
<code>BaseClient.java</code>	Java class that extends <code>EgenClient</code> class that calls the various mainframe services.

File Name	File Purpose
<code>Client.java</code>	The user interface client class. It receives a command line option to roll back or commit from the user. All invocations of the mainframe services that it makes are made by calling the <code>callService</code> method of its <code>BaseClient</code> member. First, it deletes a record and clears the JMS queue. Then, it initiates a transaction. Within the boundaries of that transaction, it adds the record and queues the key to the record on the JMS queue. It then commits or rolls back the previous operations based on the command line option. It then verifies the operation by attempting to read the record and checking the contents of the JMS queue.

The files for CICS side of the sample are as follows:

File Name	File Purpose
<code>EMPREC</code>	COBOL copybook that defines the structure of the employee record data.
<code>DPLDEMVC</code>	CICS server program that inserts a record in a VSAM file.
<code>DPLDEMVR</code>	CICS server program that reads a record in a VSAM file.
<code>DPLDEMVCU</code>	CICS server program that updates a record in a VSAM file.
<code>DPLDEMVD</code>	CICS server program that deletes a record in a VSAM file.
<code>CMPPROC</code>	Procedure used to compile and link the CICS programs.
<code>COMPILEV</code>	JCL that executes the <code>CMPPROC</code> for the programs <code>DPLDEMVC</code> , <code>DPLDEMVR</code> , <code>DPLDEMVCU</code> , and <code>DPLDEMVD</code> .
<code>JVSAMRDO</code>	RDO cards to define <code>DPLDEMVC</code> , <code>DPLDEMVR</code> , <code>DPLDEMVCU</code> , and <code>DPLDEMVD</code> programs and the <code>BEAJAMTV</code> VSAM file to the CICS region.
<code>CSDUPDCT</code>	JCL that executes <code>JVSAMRDO</code> .
<code>JAMVSAMC</code>	Delete/Define cards for the VSAM file.
<code>BLDVSAM</code>	JCL that invokes <code>IDCAMS</code> for <code>JAMVSAMC</code> .

Setting Up the Sample

To set up the transactional sample from WebLogic Server to CICS, complete the following steps.

Step 1: Enable the Services

To enable the DPL services, click **Java Adapter for Mainframe** → **Services** → **DPLService** in the left pane. Click **sampleCreateT**. Check **Enabled** and click **Apply** to enable the Local Service.

Repeat this step for each of the services:

- sampleReadT
- sampleUpdateT
- sampleDeleteT

The screenshot shows a 'Configuration' dialog box with two tabs: 'General' and 'CRM Links'. The 'General' tab is active. It contains the following fields and controls:

- Service Name:** sampleCreateT
- CICS Program Name:** DPLDEMVC
- Alternate Transaction Id:** (empty text box)
- Timeout:** 30000 (in milliseconds)
- Input Schema:** (empty text box)
- Output Schema:** (empty text box)
- Enabled:** ☒
- Buttons:** 'Apply' and 'Reset'

Step 2: Set the Environment

On the machine where the Gateway is installed, set the environment by completing the following step:

1. From a command prompt, execute the following command to set the environment:

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

```
.. ./setExamplesEnv.sh
```

The following message will display:

“Your environment has been set.”

Step 4: Complete Mainframe Tasks

On the machine with the CICS region:

1. Create a Partitioned Data Set (PDS) to store the source and JCL for this sample.
2. From the machine where the Gateway was installed, FTP the following files into the PDS that you created:
 - EMPREC
 - DPLDEMVC
 - DPLDEMVR
 - DPLDEMVCU
 - DPLDEMVD
 - CMPPROC
 - COMPILEV
 - JVSAMRDO
 - CSDUPDCT
 - JAMVSAMC
 - BLDVSAM
3. In the procedure `CMPPROC`, do not set `LNKLIB`, `PDSSRC`, and `PROG`. The values are supplied by the `COMPILEV` JCL that will execute `COMPROC`. Do set `INDEX`, `COMPHLQ`, and `COMPHL2`. You may need to change `OUTC` and the unit of `WORK`.
4. In the `COMPILEV` JCL, make the following changes:
 - 1 Change the `JOB` statement.
 - 1 Change the `JCLLIB` `ORDER` and the `SOURCE` to the PDS you created.
 - 1 Change the `LOADLIB` to a load lib for your CICS region.
5. Submit the `COMPILEV` JCL. Make sure that the condition code is 0.

6. In the `JAMVSAMC` member, make the following changes:
 - 1 Change all occurrences of `YOURHLQ.JAM.VSAMFILE` to the name you choose for the VSAM file used in this sample.
 - 1 Change `YOURVOL` to an appropriate volume ID.
7. In the `BLDVSAM JCL`, make the following changes:
 - 1 Change the `JOB` statement.
 - 1 Change `YOURHLQ.JAM.WORKFILE` to the PDS you created in (1).
8. Submit the `BLDVSAM JCL`. Verify the results. One data set should be created with no extension, one data set created with `DATA` as the extension, and one data set with the `INDEX` extension.

Note: A condition code of 8 is acceptable on the `DELETE` step; however, the condition code should be 0 on the `DEFINE` step.
9. In the RDO script, `JVSAMRDO`, make the following changes:
 - 1 Change the name of the `LIST` to a valid list name for your CICS region.
 - 1 You can also change the `GROUP` name, but make sure it is changed everywhere.
 - 1 In the `DEFINE` statement for the VSAM file `BEAJAMTV`, change the `DSNAME` to match the name for the VSAM file set in `JAMVSAMC`. The value to which `YOURHLQ.JAM.VSAMFILE` is changed must be the same in both `JVSAMRDO` and `JAMVSAMC`.
10. In the `CSDUPDCT JCL`, make the following changes:
 - 1 Change the `JOB` statement.
 - 1 Change the `STEPLIB`.
 - 1 Change the `DFHCSD`.
 - 1 Change the `SYSIN DSN`.
11. Submit the `CSDUPDCT JCL`.

You may get a warning on the `DELETE` steps, because the programs and file were not previously defined. Make sure the condition code is not higher than 4.
12. Log on to your CICS region.
13. Install the iWay JAM sample.

To install the iWay JAM sample, type the following command at the command prompt:

```
CEDA INSTALL GROUP(JAMEXMPL)
```
14. Verify the CICS sample programs.

To verify the CICS sample programs, type the following command at the command prompt:

```
CEMT INQUIRE PROG (DPLDEMVC)
```

Repeat for DPLDEMVR, DPLDEMVCU, and DPLDEMVD.

Also make sure that the VSAM file is open:

```
CEMT INQUIRE FILE (BEAJAMTV)
```

Running the Sample

To run the sample, type the following command at the command prompt:

```
java examples.transactional.CICS.outbound.gateway.Client [-m hostname] [-p port] [-r]
```

In this command, the following definitions apply:

- n The first pair of command line switches are optional, and represent the `hostname` and `port` for a remote machine on which the iWay JAM Gateway is running in WebLogic Server. If you are running the client on the same machine as the Gateway and the WebLogic Server port is 7001, then these options are not required.
- n The third command line switch, which is also optional, indicates whether a rollback will be done or not. If `-r` is present at the command prompt, the creation of the record on the database will be rolled back.

The following command is an example of a command that you might enter:

```
java examples.transactional.CICS.outbound.gateway.Client -r
```

CHAPTER 4

Using the Explicit APPC Samples

Topics:

- About the Explicit APPC Samples
- Roadmap for the Samples
- Using the Samples

The explicit APPC sample demonstrates how iWay Java Adapter for Mainframe (iWay JAM) integrates the WebLogic applications with batch MVS COBOL applications.

About the Explicit APPC Samples

The following section provides an overview of the explicit APPC sample: Batch MVS COBOL Client to WebLogic EJB Sample. A detailed description of how the sample works and instructions for running the sample are provided in *Using the Samples* on page 4-3.

Batch MVS COBOL Client to WebLogic EJB Sample

This sample demonstrates the functional capability of iWay Java Adapter for Mainframe (iWay JAM) to invoke the services of an Enterprise Java Bean (EJB) deployed with WebLogic Server from a mainframe application, specifically a batch MVS client using explicit APPC. This invocation is facilitated by the EJB API assembler interface that is delivered as part of this sample.

The MVS COBOL client receives a string of text as input. In this sample, the business function of the EJB is to convert the string of text to uppercase and return the result to the client. The string displays in the WebLogic Server log by the EJB before and after conversion. The client displays the result on the system output device.

Transactional Batch MVS COBOL Client to WebLogic EJB Sample

This sample demonstrates the capability of iWay JAM to invoke the services of an Enterprise JavaBean (EJB) within the scope of a transaction. In this case the Client is an APPC batch job, which uses IBM Resource Recovery Services (RRS) calls to delimit the transaction. The RRS calls are collected within a "C" interface, APPCCRM, included with the sample, to demonstrate a possible way to organize these calls in a reusable fashion.

The EJB method TOUPPER is invoked, as in the non-transactional case. This EJB method does no transactional work; it may be adapted to use transactional resources if desired.

This sample is for reference only. Full end-to-end deployment details are not yet available.

Roadmap for the Samples

To run the explicit APPC sample, follow the roadmap listed below:

1. Verify prerequisite tasks.

For a listing of prerequisite tasks, see Chapter 1, *iWay Java Adapter for Mainframe Samples Overview*.

2. Prepare to use the explicit APPC sample.

- a. Start the CRM.
- b. Set Logical Unit VTAM definitions.
- c. Update the iWay JAM Configuration File

- d. Start the `examples` domain
 - e. Configure the iWay JAM Gateway
 3. Set up the Sample
 - a. Configure Services
 - b. Set the Environment
 - c. Generate and Build Source (Optional)
 - d. Complete Mainframe Tasks
 4. Run the Sample

Using the Samples

After you have completed the tasks described in the [“Before You Run the Samples”](#) section, you are ready to use the explicit APPC sample. Information about how to use the explicit APPC sample is presented in the following sections:

- Preparing to Use the Explicit APPC Sample
- Using the Batch MVS COBOL Client to WebLogic EJB Sample
 - Understanding How the Sample Works
 - Setting up the Sample
 - Running the Sample

Preparing to Use the Explicit APPC Sample

To use the Explicit APPC Sample, you must complete the following steps.

Step 1: Start the CRM

Before starting the iWay JAM Gateway, start the CRM. The CRM must be configured with certain parameter values at startup. These parameter values include:

- The address of the machine on which the CRM is running
- The port on which the CRM listens
- The name the Gateway will use to refer to the CRM

For running the samples, you must set the machine address and port. The values that you set for the machine address and port when the CRM is started, must agree with the values that you set for the CRM in the WebLogic Administration Console for the samples CRM. The name of the CRM that is preconfigured for running all of the samples is CRM1. Use this name when the CRM is started to run any of the samples.

The way you start the CRM depends on whether the CRM will be started under a Unix or MVS system. On Unix, start the CRM using a shell script. On MVS, start the CRM using JCL. For more information, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*.

Step 2: Set Logical Unit VTAM Definitions

This sample uses a Batch MVS client and requires a partner Logical Unit to be defined. Extra steps are required to make the necessary VTAM definitions for this sample. For information about Logical Unit definition and VTAM definition, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*.

1. Define a VTAM APPC Logical Unit to be used by the `EJBAPI` interface. The `EJBAPI` establishes an APPC conversation with the CRM. It must have access to an Logical Unit for this purpose.
2. Configure APPC to use the new Logical Unit. You must add an `LUADD` statement to the `APPCPMxx PARMLIB` member. See the example in `VTAMDEFINITION`.

The `EJBAPI` does not set the name of the Logical Unit it uses, but relies on the default APPC Logical Unit. As a result, the `LUADD` statement that is added to the `APPCPMxx PARMLIB` member should be the last in the member with the `BASE` attribute. Also, the `ACBNAME` must match the `ACBNAME` for the Logical Unit defined for the `EJBAPI` interface.

3. Define an APPC `SYMDEST`. Use the APPC administration facility to define the `SYMDEST`. See `VTAMDEFINITION` for an example.

The `Partner LU` must match the Logical Unit defined for the CRM. The `TP Name` must be the name configured for the EJB in the iWay JAM configuration. For this example, the name is `TOUPPER`.

Step 3: Update the iWay JAM Configuration File

On the machine where the example domain is installed, switch to your examples domain by copying `jamconfig_BATCH.xml` to `jamconfig.xml`.

Step 4: Start the examples Domain

From the command prompt, execute the following command from the same directory to start the `examples` domain:

- For Microsoft Windows:
`startExamplesServer.cmd`
- For Unix:
`../startExamplesServer.sh`

Step 5: Configure the iWay JAM Gateway

Most configuration tasks were preconfigured or were completed during the installation process by the installer program. For additional information about configuring iWay JAM, see the *iWay Java Adapter for Mainframe Configuration and Administration Guide*. Make the following configuration changes for the IMS Installation Verification Sample to run on your system. These changes can be made in the WebLogic Administration Console in the following way.

1. From your browser, open the WebLogic Administration Console using the following address:

`http://hostname:7001/console`

In this address, the following definitions apply:

- `hostname` is the address of the machine where WebLogic Server is running.
- `7001` is the port for WebLogic Server that has been configured for the `examples` domain.

2. When prompted, supply the following user and password information:

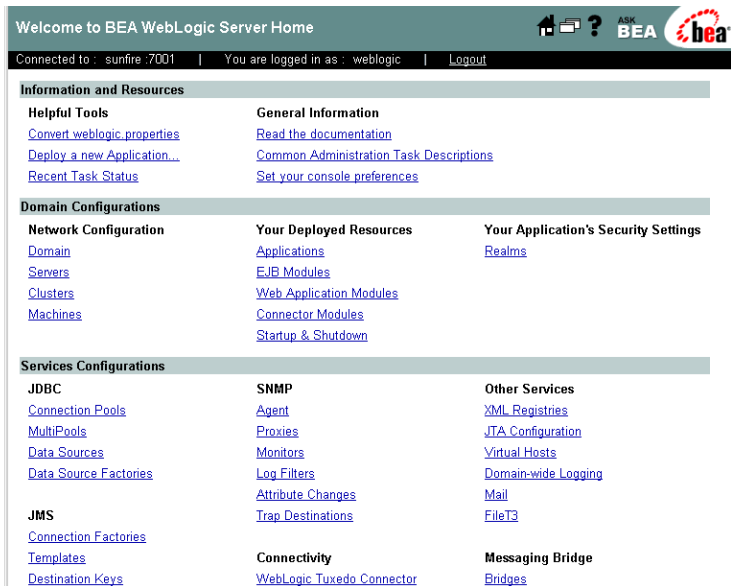
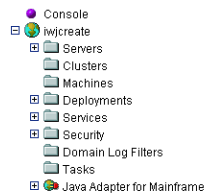
- user: `system`

This user name cannot be changed.

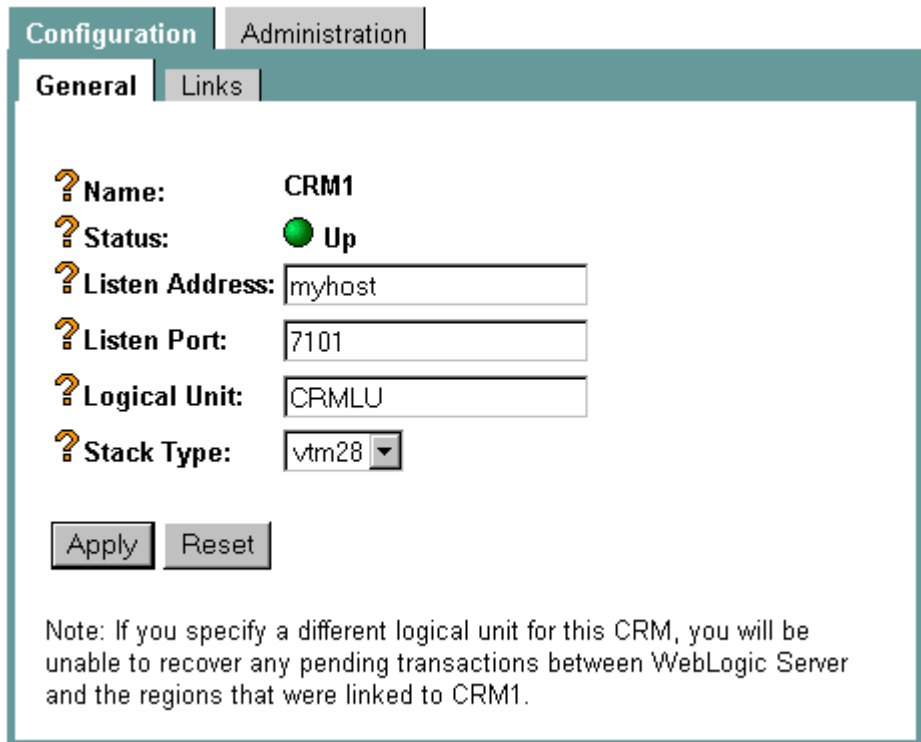
- password: `security`

To change the password, see the BEA WebLogic Server documentation.

The WebLogic Administration Console displays.



3. To configure the CRM to the iWay JAM Gateway, complete the following steps:
 - a. In the left pane, click on **Java Adapter for Mainframe** → **JAM Components** → **CRMs**. In the right pane, click **CRM1**. On the **General** tab, set the following fields to correspond with your system. Click **Apply**. When the CRM is active, **Status** turns from red to green.



Configuration Administration

General Links

? **Name:** CRM1

? **Status:** ☒ Up

? **Listen Address:** myhost

? **Listen Port:** 7101

? **Logical Unit:** CRMLU

? **Stack Type:** vtm28

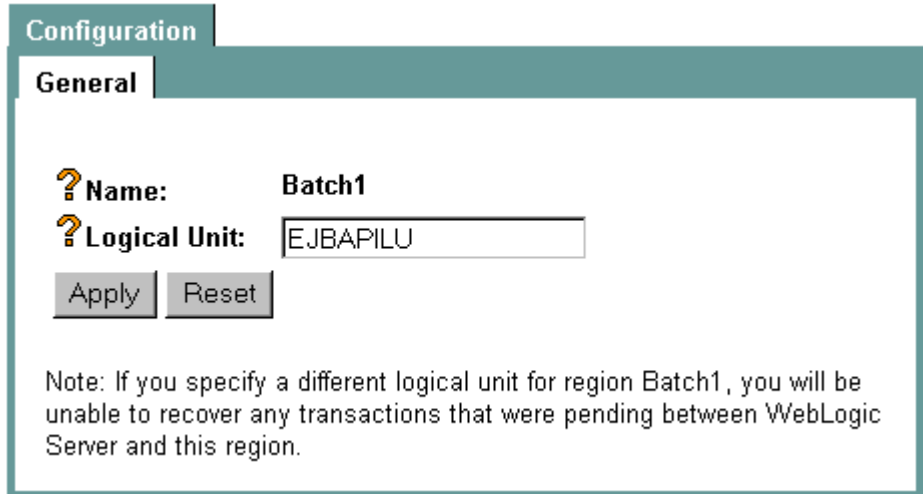
Apply Reset

Note: If you specify a different logical unit for this CRM, you will be unable to recover any pending transactions between WebLogic Server and the regions that were linked to CRM1.

Field	Field Description
Listen Address	The address of the machine where the CRM is installed and running. This address must match the address set in the CRM startup JCL or script.
Listen Port	The port for the CRM. This entry must match the port set in the CRM startup JCL or script.
Logical Unit	The name of the Logical Unit defined for the CRM.
Stack Type	The stack type.

- b. To configure the Batch region, click **Java Adapter for Mainframe** → **Regions** → **BATCH Regions** in the left pane. In the right pane, click on **Batch1** and enter the name of the **Logical Unit** that you set in Step 2. Click **Apply** to set the Logical Unit.

Note: This Logical Unit is not the Logical Unit for the CRM in (3a).



The screenshot shows a configuration window with a teal header bar. The 'Configuration' tab is selected, and the 'General' sub-tab is active. The 'Name' field is labeled with a question mark icon and contains the text 'Batch1'. The 'Logical Unit' field is also labeled with a question mark icon and contains the text 'EJBAPILU'. Below these fields are two buttons: 'Apply' and 'Reset'. At the bottom of the configuration area, there is a note: 'Note: If you specify a different logical unit for region Batch1 , you will be unable to recover any transactions that were pending between WebLogic Server and this region.'

- c. Click **Batch Regions** at the top of the pane. In the new window, click **CRM1Batch**. On the **Links** tab, check **Deployed** and click **Apply**.

The screenshot shows a configuration window titled "Configuration Administration" with a "Links" tab selected. The configuration details for a link named "CRM1Batch" are as follows:

? Name:	CRM1Batch
? Status:	Down
? Region Name:	Batch1 (BatchRegion) ▼
? Logmode Name:	SMSNA100
? Maximum Sessions:	4
? Minimum Winner Sessions:	2
? Security Level:	Local ▼
? Default User-Id:	
? Deployed:	☒

At the bottom of the configuration area are two buttons: "Apply" and "Reset".

Note: By modifying the parameters of this link you will be unable to recover any transactions that were pending on this link between WebLogic Server and region Batch1.

- d. In the left pane, click **Gateways**. Click **JAM5.1** in the right pane. On the **General** tab, check **Deployed** and click **Apply**.

The screenshot shows the 'Configuration' window with the 'Administration' tab selected. Inside, the 'General' sub-tab is active. The configuration fields are as follows:

Field	Value
Name	JAM5.1
Status	Down (indicated by a red circle icon)
Target WebLogic Server	exampleserver (dropdown menu)
Target CRM	CRM1 (dropdown menu)
Deployed	☐

Buttons: Apply, Reset

Note: By modifying the Target CRM parameter of this gateway you will be unable to recover any transactions that were pending for gateway JAM5.1.

4. To start the iWay JAM Gateway, select the **Administration** tab → **Start/Stop** tab. Click **Start** to start the Gateway.

The screenshot shows the 'Administration' window with the 'Start / Stop' sub-tab selected. The title is 'Start / Stop Gateway'. The configuration fields are:

Field	Value
Name	JAM5.1
Status	Down (indicated by a red circle icon)

Buttons: Start, Stop

If the Gateway is running, **Status** changes to green in the WebLogic Administration Console and the following message appears in the WebLogic Server log:

"JAM Gateway ready for use. Current link status: up(1)."

Using the Batch MVS COBOL Client to WebLogic EJB Sample

After completing the steps in *Preparing to Use the Explicit APPC Sample* on page 4-3 section, you are ready to set up and run the explicit APPC sample: batch MVS COBOL client to WebLogic EJB sample.

Understanding How the Sample Works

This sample demonstrates the functional capability of iWay JAM to invoke a service offered by an EJB deployed in WebLogic Server from a mainframe application, specifically a batch MVS COBOL client using explicit APPC. This invocation is facilitated by the `EJBAPI` assembler interface that is delivered as part of the sample.

Understanding the Sample Configuration

When using iWay JAM to integrate with CICS or IMS applications, you must define an Logical Unit for the CRM. This sample consists of a batch program using explicit APPC and requires a partner Logical Unit to be defined. A partner Logical Unit must be defined for use by the `EJBAPI`. Note that the `EJBAPI` does not set the name of the Logical Unit that it uses. It relies on the default APPC Logical Unit. Therefore, the `LUADD` statement that is added to the `APPCPMxx` `PARMLIB` member, corresponding to the Logical Unit defined for the `EJBAPI` interface to use, must be the last in the member with the `BASE` attribute.

The `EJBAPI` also makes use of a symbolic destination or `SYMDEST` to establish an APPC conversation with the CRM. The `Partner LU` in the `SYMDEST` must be the Logical Unit defined for the CRM. The `TP Name` must be the name of the `EJBExport` element in the iWay JAM configuration, in this case `TOUPPER`.

In the iWay JAM configuration, an `EJBExport` element is defined with the name that matches the `TP Name` in the `SYMDEST` for the `EJBAPI`, `TOUPPER`. The JNDI-name attribute of this `EJBExport` element `jam.ToupperServer` is the value of the JNDI-name element in the WebLogic deployment descriptor `weblogic-ejb-jar.xml` for the `ToupperServer` EJB.

The eGen Application Code Generator generates deployment descriptors when it generates the code for EJBs. However, to avoid name collisions that can occur when multiple EJBs are generated from a single eGen script, generated deployment descriptors are always given names that contain the stem name of the EJB that is being generated. In this sample, the generated deployment descriptors are named `ToupperServer-jar.xml` and `wl-ToupperServer-jar.xml`. Before these deployment descriptors can be used for an actual deployment in WebLogic Server, they have to be renamed `ejb-jar.xml` and `weblogic-ejb-jar.xml`. Because the generated EJB `ToupperServerBean` is extended in this sample, the value of the `ejb-class` element in `ejb-jar.xml` must also be manually changed to the name of the extension class that contains the business logic, `ExtToupperServerBean`.

Understanding the Sample Programming

The programming for this sample is described in the following sections.

WebLogic Application

Five classes compose the WebLogic side of this sample application:

- Chardata
- ToupperServer
- ToupperServerBean
- ToupperServerHome
- ExtToupperServerBean

`Chardata` is a `DataView` class that is generated by the eGen Application Code Generator. The data member in the `Chardata` class corresponds to the data field in the `CHARDATA` copybook. The `Chardata` class is responsible for all data translation between the mainframe format of the data and the Java format of the data

`ToupperServer`, `ToupperServerBean`, and `ToupperServerHome` classes are generated by the eGen Application Code Generator. `ToupperServer` is a remote interface that contains the definition of a single method, `dispatch`. `dispatch` is the essential method for server EJBs that are used in iWay JAM applications. This method is messaged by the Gateway when a mainframe client makes a request of the corresponding service. `ToupperServerHome` is a standard home interface for a stateless session bean. It contains the definition of a `create` method that returns a `ToupperServer` object to the caller. `ToupperServerBean` extends `EgenServerBean`. `ToupperServerBean` contains an implementation of the `dispatch` method that is a wrapper for the `toupper` method. The implementation of the `toupper` method that is given in `ToupperServerBean` does not perform actual business logic. As it is defined in the eGen script that generates `ToupperServerBean`, the `toupper` method only receives a `Chardata` object and returns a `Chardata` object. To actually do any business logic, the `ToupperServerBean` must be extended and the `toupper` method overwritten.

The extension of the `ToupperServerBean` class that is included with this sample is called `ExtToupperServerBean`. `ExtToupperServerBean` contains an implementation of the `toupper` method containing the business logic. The string data member in the received `Chardata` object is written to the WebLogic log. The string is converted to uppercase, and the resulting string is written to the WebLogic Server log and returned to the client.

MVS Program

`WLCLIENT` is a simple COBOL batch client program that makes a synchronous request of a service offered by an EJB deployed in WebLogic Server by calling the `EJBAPI` assembler interface. The request consists of the string, "This is a string of text." The EJB will convert the string to uppercase and return it to the client. The response string that is returned to `WLCLIENT` is displayed in `SYSOUT`.

`EJBAPI` is an assembler interface that is called for use by programs to invoke the services of an EJB deployed in a WebLogic Server instance. `EJBAPI` establishes an APPC conversation with the CRM. The `EJBAPI` sends the request data to the CRM that then forwards the information to the iWay JAM Gateway. The `EJBAPI` issues a receive to retrieve the response data.

Reference: Sample Files

The following table lists the samples files and their purpose:

File Name	File Purpose
chardata.cpy	COBOL copybook that defines the structure of the string mainframe data.
chardata.egen	eGen script that generates the <code>Chardata.java</code> <code>DataView</code> class.
Chardata.java	<code>DataView</code> class that corresponds to the <code>chardata.cpy</code> COBOL copybook.
toupperServer.egen	eGen script that generates the <code>Chardata.java</code> <code>DataView</code> class, the <code>ToupperServer.java</code> , <code>ToupperServerBean.java</code> , <code>ToupperServerHome.java</code> bean classes, and the <code>ToupperServer-jar.xml</code> and <code>wl-ToupperServer-jar.xml</code> deployment descriptors.
ToupperServer.java	EJB remote interface generated by the eGen utility.
ToupperServerBean.java	EJB generated by the eGen utility.
ToupperServerHome.java	EJB home interface generated by the eGen utility.
ExtToupperServerBean.java	EJB that extends <code>ToupperServerBean</code> . The business logic of servicing the requests from the mainframe is actually implemented in this class.
ToupperServer-jar.xml	Deployment descriptor generated by <code>toupperServer.egen</code> .
wl-ToupperServer-jar.xml	WebLogic deployment descriptor generated by <code>toupperServer.egen</code> .

File Name	File Purpose
ejb-jar.xml	ToupperServer-jar.xml deployment descriptor that has been modified and renamed for inclusion in the JAM_ToupperServer.jar.
weblogic-ejb-jar.xml	wl-ToupperServer-jar.xml WebLogic deployment descriptor renamed for inclusion in the JAM_ToupperServer.jar.

The files for the mainframe side of the sample are listed and described in the following table: installed in the following directory:

File Name	File Purpose
ASEJBAPI	JCL that assembles the WebLogic EJB API interface module <code>EJBAPI</code> .
CLCLIENT	JCL to compile and link the batch client <code>WLCLIENT</code> .
EXCLIENT	JCL to execute the batch client <code>WLCLIENT</code> .
EJBAPI	The WebLogic EJB API interface module.
VTAMDEFINITION	Contains sample Logical Unit definitions, a sample <code>SIDEINFO</code> definition, and a sample <code>SYMDEST</code> definition.
WLCLIENT	MVS batch COBOL client that makes a request to a service provided by an EJB through iWay JAM to convert a string to uppercase. The response is displayed in <code>SYSOUT</code> .

Setting Up the Sample

To set up the batch MVS COBOL client to WebLogic EJB sample, complete the following steps.

Step 1: Enable the Service

1. To enable the Exported EJB, click **Java Adapter for Mainframe** → **Exports** → **ExportedEJBs** in the left pane. Click **TOUPPER**. Check **Local Service Enabled**. Click **Apply** to enable the Local Service.

The image shows a 'Configuration' dialog box with a 'General' tab. It contains the following fields and controls:

- Service Name:** TOUPPER
- JNDI Name:** jam.ToupperServer
- Timeout:** 30000 (in milliseconds)
- Local Service Enabled:** ☒
- Buttons:** Apply, Reset

Step 2: Set the Environment

On the machine where the Gateway is installed, set the environment in the following way:

1. From a command prompt, execute the following command to set the environment:

- For Microsoft Windows:

setExamplesEnv.cmd

- For Unix:

./setExamplesEnv.sh

The following message will display:

"Your environment has been set."

Step 3: Complete Mainframe Tasks

On the machine where the batch client is to run:

1. Allocate a Partitioned Data Set (PDS) that will store the source and JCL for the sample. Allocate a PDS for the objects that are created for this sample. Both of these PDSs should be allocated with Record Format: FB and Record Length: 80.
2. Allocate a PDS for the executable that is built in this sample. This PDS should be allocated with Record Format: U and Record Length: 0.

3. From the machine where the Gateway was installed, FTP the following files to the source PDS that you created:
 - ASEJBAPI
 - CLCLIENT
 - EJBAPI
 - EXCLIENT
 - WLCLIENT
4. In the `ASEJBAPI` JCL, make the following changes:
 - a. Change the `JOB` statement.
 - b. Change the `YOURHLQ.OBJECT` to the PDS that you allocated for objects.
 - c. Change the `YOURHLQ.SOURCE` to the PDS that you allocated for source.
5. Submit the `ASEJBAPI` JCL. Make sure that the condition code is 0.
6. In the `CLCLIENT` JCL, make the following changes:
 - a. Change the `JOB` statement.
 - b. Change the `YOURHLQ.OBJECT` to the PDS that you allocated for objects.
 - c. Change the `YOURHLQ.SOURCE` to the PDS that you allocated for source.
 - d. Change the `YOURHLQ.LOAD` to the PDS that you allocated for executables.
 - e. Change `STEPLIB, SYSLIB, CSSLIB`, if necessary.
7. Submit the `CLCLIENT` JCL. Make sure that all the condition codes are 0.
8. In the `EXCLIENT` JCL, make the following changes:
 - a. Change the `JOB` statement.
 - b. Change the `YOURHLQ.LOAD` to the PDS that you allocated for executables.
 - c. Do not submit this JCL until you are ready to run the client.

Running the Sample

To run the sample, submit the `EXCLIENT` JCL. Make sure that the condition code is 0.

You will see the request string in the WebLogic Server log before and after the conversion to uppercase. The converted string will also be displayed in `SYSOUT`.

Using the Transactional Batch MVS COBOL Client to WebLogic EJB Sample

The sample contains the following files that can be used to make a transactional batch application, which calls TOUPPER.

Sample File	Description
APPCCRM	The RRS/APPC interface program.
APPCMVS	The APPCMVS definition for protected conversation LU.
CCOMP	The APPCCRM compile JCL.
CLNTEXEC	The Client execute JCL.
COBCOMP	The CRMCLNT compile JCL.
CRMCLNT	The Transactional Batch client program.
VTAMDEF	The VTAM definition for protected conversation LU.

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. You can contact us through the following methods:

Mail: Documentation Services - Customer Support
Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898

Fax: (212) 967-0460

E-mail: books_info@ibi.com

Web form: <http://www.informationbuilders.com/bookstore/derf.html>

Name: _____

Company: _____

Address: _____

Telephone: _____ Date: _____

E-mail: _____

Comments:

Reader Comments