BEA WebLogic Java Adapter for Mainframe™

Samples Guide

Release 5.1
Document Date: August 2002

BEA WebLogic Java Adapter for Mainframe Samples Guide

Part Number	Date	Software Version
N/A	August 2002	5.1

Contents

4. Using the Explicit APPC Sample

Index

CHAPTER

1 BEA WebLogic Java Adapter for Mainframe Samples Overview

BEA WebLogic Java Adapter for Mainframe (WebLogic JAM) is designed to integrate applications running in a WebLogic Server environment with applications running in a mainframe environment. WebLogic JAM includes complete samples that are installed as part of the product installation. These samples are provided to demonstrate how WebLogic JAM integrates WebLogic and mainframe applications. This guide explains how to run the samples that are included with WebLogic JAM.

The following topics are described in this section:

- About the WebLogic JAM Samples
- How to Use the Samples

About the WebLogic JAM Samples

The samples included with WebLogic JAM allow you to see different capabilities of WebLogic 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 WebLogic JAM software to:

- Verify that the product was installed correctly
- Demonstrate the way WebLogic 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 WebLogic JAM:

- Verification Samples
- Programming Samples

Verification Samples

WebLogic 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 WebLogic JAM installation as well as demonstrate WebLogic JAM integration in a WebLogic and mainframe environment. These samples are described in the *BEA WebLogic Java Adapter for Mainframe Installation Guide*.

Programming Samples

The samples that are categorized as programming samples demonstrate how WebLogic 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 WebLogic JAM from IMS to a Java application running under 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.
- CICS Samples
  - CICS Application to WebLogic Server Sample EJBs

    This sample demonstrates the functional capability provided by WebLogic 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 Sample
  - 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.

About the Samples User

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

- Systems Programmer
- Mainframe Application Programmer
- Java Application Programmer

For a description of user roles, see the *BEA WebLogic Java Adapter for Mainframe Home Page*.

How to Use the Samples

While the WebLogic 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 WebLogic 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 WebLogic JAM installation includes a directory named `config`. This `config` 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 WebLogic 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 WebLogic JAM.

   For information about installing WebLogic JAM, see the *BEA WebLogic 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 *BEA WebLogic 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 WebLogic JAM, see the *BEA WebLogic 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 WebLogic JAM, see the *BEA WebLogic Java Adapter for Mainframe Configuration and Administration Guide.*

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

- Using the IMS Samples
- Using the CICS Samples
- Using the Explicit APPC Sample

CHAPTER

2 Using the IMS Samples

The IMS samples demonstrate how BEA WebLogic Java Adapter for Mainframe (WebLogic JAM) integrates WebLogic applications with IMS applications on a mainframe. This section provides the following information:

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

About the IMS Samples

The following section provides a brief overview of each of the IMS 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.

IMS Application to WebLogic Server JMS Topic

This sample demonstrates an asynchronous call through WebLogic 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic 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`.

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 the "Before You Run the Samples" section.

2. Prepare to use the IMS sample.

   a. Start the CRM.

   b. Update the WebLogic JAM configuration file.

   c. Start the `examples` domain.

   d. Configure the WebLogic 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 the "Before You Run the Samples" section, you are ready to use the sample. Information about how to use the IMS samples is presented in the following sections:

- Preparing to Use the IMS Samples
- Using the IMS Application to WebLogic Server JMS Topic
  - Understanding How the Sample Works
  - Setting up the Sample
  - Running the Sample
- Using the Java Client to IMS Sample Application
  - Understanding How the Sample Works
  - Setting up the Sample
  - Running the Sample
- Using the Transactional Sample from WebLogic Server to IMS
  - Understanding How the Sample Works
  - Setting up the Sample
  - Running 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 WebLogic 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.

Starting the CRM on z/OS or OS/390 Unix

On z/OS or OS/390 Unix, you may use a script to start the CRM. Scripts are installed with the Gateway in the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/crm/unix` directory. The script, `crm.env`, appends the necessary values to your environment variables. The script, `startcrm.sh`, starts the CRM. To use these scripts, complete the following steps:

1. FTP the following two scripts to the directory from which the CRM will run:
   - `crm.env`
   - `startcrm.sh`
2. Edit `crm.env`. Supply the correct values for the `APPDIR` and `CRMDIR` variables. `APPDIR` is the directory from which the CRM will run. `CRMDIR` is the CRM installation directory.

3. Edit `startcrm.sh`. To use a different port than the default port, `7101`, change the port number. However, if you change the port number, make sure to change it in the corresponding field in the WebLogic Administration Console CRM1 pane. You do not need to change the address because the script will run on the machine where the CRM is installed.

   Note: BEA recommends that you do not change the CRM name from CRM1. This name for the CRM is preconfigured for all of the samples.

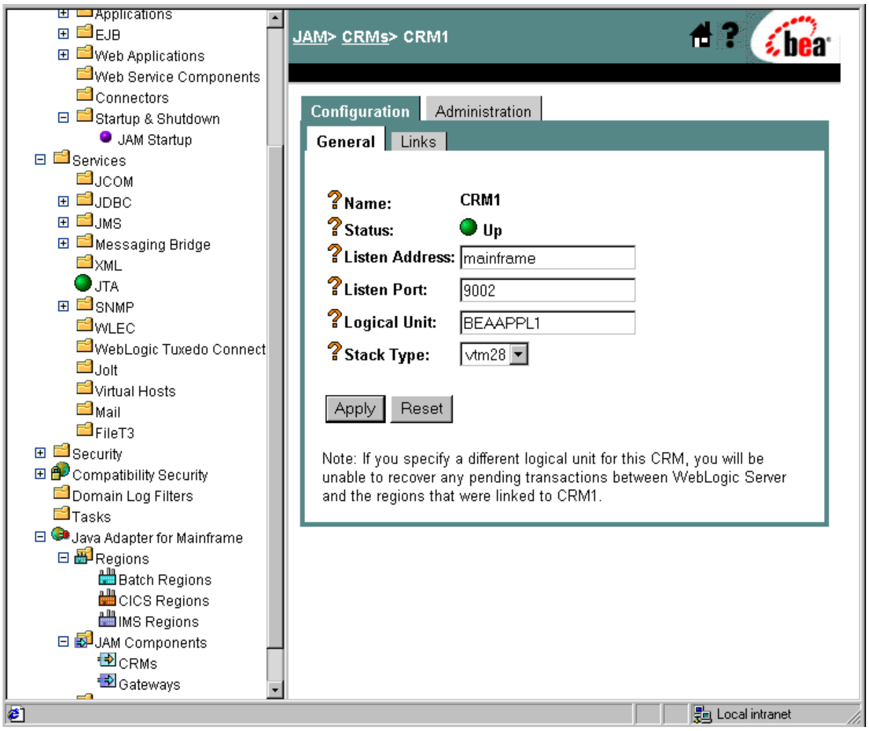
4. Execute the `startcrm.sh` script:

   ```
   . ./startcrm.sh
   ```

Compare Figure 2-1 with the script in Listing 2-1. Notice how the parameters in the script correspond to the fields in the WebLogic Administration Console. The script illustrates the values for `startcrm.sh` script parameters for running the samples.

Note: The port number is `7101`. You can change the port number; however, if the port number is changed make sure to change it in the corresponding field in the Gateway configuration CRM1 pane of the WebLogic Administration Console.

Figure 2-1 Fields for the CRM



Name: CRM1
Status: ● Up
Listen Address: mainframe
Listen Port: 9002
Logical Unit: BEAAPPL1
Stack Type: vtm28

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.

Listing 2-1 Command to Run the CRM

```
$CRMDIR/bin/CRM //127.0.0.1:7101 CRM1 < /dev/null > std.out
2>std.err &
```

Starting the CRM on z/OS or OS/390 MVS

On z/OS or OS/390 MVS, start the CRM by submitting the `CRMSTART` JCL that is installed with the CRM. The `CRMSTART` JCL must be modified for your environment. For information about modifying the `CRMSTART` JCL, see the *BEA WebLogic Java Adapter for Mainframe Configuration and Administration Guide*.

As you modify the `CRMSTART` JCL, make sure that you note the following parameters in the value of the `STARTCMD` parameter in the JCL. These parameters correspond to fields in the WebLogic Administration Console. These values must be the same in the JCL and in the WebLogic Administration Console.

- The machine address where the CRM will run
- The port number on which the CRM will listen
- The name by which the Gateway will refer to the CRM

Compare Figure 2-1 with the JCL in Listing 2-2. Notice how the parameters in the JCL correspond to the fields in the WebLogic Administration Console. The JCL illustrates the values for `STARTCMD` parameters for running the samples.

- The machine where the CRM will run is `myhost` in this sample. You must replace `myhost` with the hostname or IP address of your mainframe to allow IP-based communication from the Windows or Unix machine where WebLogic Server is running. You may verify this parameter with the `ping` command on Windows or Unix.
- The port number is `7101`. You can change the port number; however, if you change the port number, make sure to change it in the corresponding field in the Gateway configuration CRM1 pane of the WebLogic Administration Console.

Note: BEA recommends that you do not change the CRM name from CRM1, because this name for the CRM is preconfigured for all of the samples.

Listing 2-2 The `STARTCMD` parameter in the `CRMSTART` JCL

```
//  SET STARTCMD='"//myhost:7101" CRM1'
```

Step 2: Update the WebLogic JAM Configuration File

On the machine where the Gateway is installed, update the WebLogic JAM configuration file from the command prompt by completing the following steps:

1. Locate the `jamconfig_IMS.xml` file under the following directory:

   ```
   <BEA_HOME>/<JAM_INSTALL_DIR>/config/examples
   ```

2. Copy `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 WebLogic JAM Gateway

Most configuration tasks are preconfigured or completed during the installation process by the installer program. For additional information about configuring WebLogic JAM, see the *BEA WebLogic 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. 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.

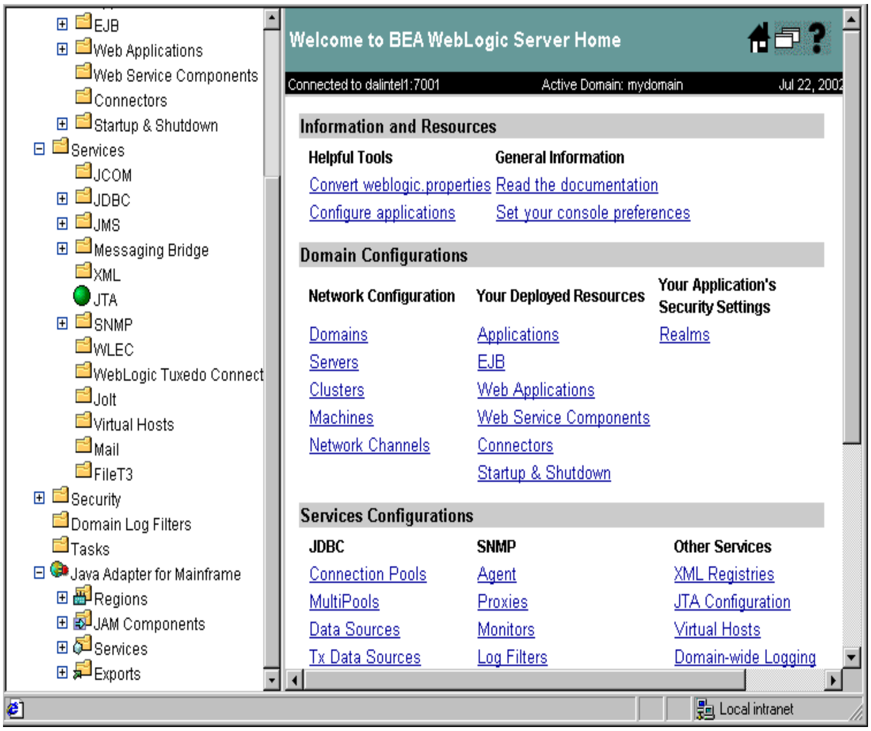
   `7001` is the port for WebLogic Server that has been configured for the `examples` domain.

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.



EJB
Web Applications
Web Service Components
Connectors
Startup & Shutdown
Services
JCOM
JDBC
JMS
Messaging Bridge
XML
JTA
SNMP
WLEC
WebLogic Tuxedo Connect
Jolt
Virtual Hosts
Mail
FileT3
Security
Domain Log Filters
Tasks
Java Adapter for Mainframe
Regions
JAM Components
Services
Exports

Welcome to BEA WebLogic Server Home

Connected to dalintel1:7001 Active Domain: mydomain Jul 22, 2002

Information and Resources

Helpful Tools	General Information
Convert weblogic.properties	Read the documentation
Configure applications	Set your console preferences

Domain Configurations

Network Configuration	Your Deployed Resources	Your Application's Security Settings
Domains	Applications	Realms
Servers	EJB	
Clusters	Web Applications	
Machines	Web Service Components	
Network Channels	Connectors	
	Startup & Shutdown	

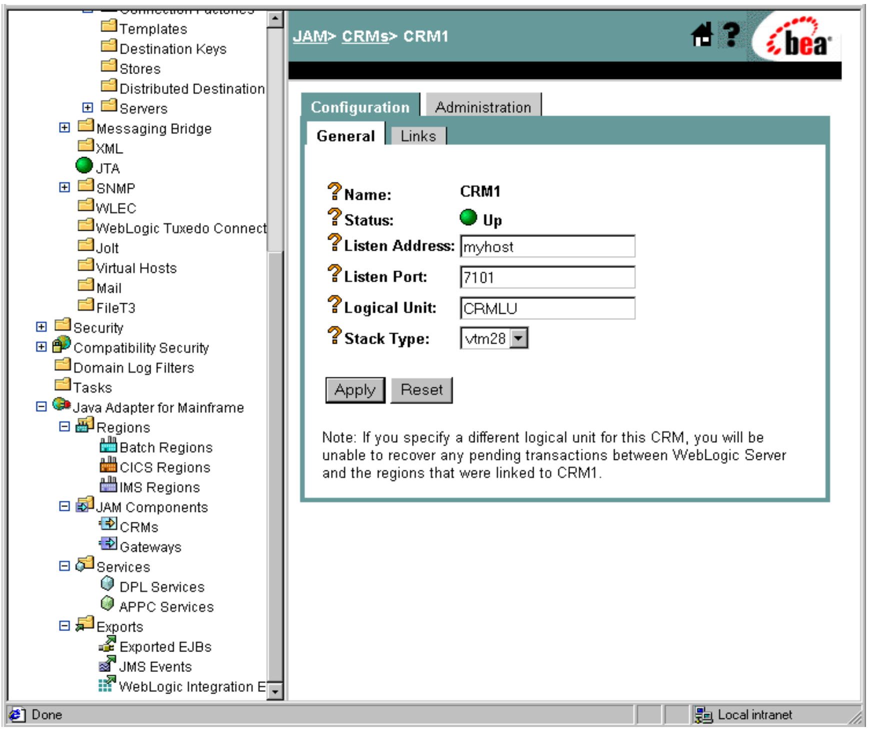
Services Configurations

JDBC	SNMP	Other Services
Connection Pools	Agent	XML Registries
MultiPools	Proxies	JTA Configuration
Data Sources	Monitors	Virtual Hosts
Tx Data Sources	Log Filters	Domain-wide Logging

Local intranet

3. To configure the CRM to the WebLogic 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.



Name: CRM1
Status: Up
Listen Address: myhost
Listen Port: 7101
Logical Unit: CRMLU
Stack Type: vtm28

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.

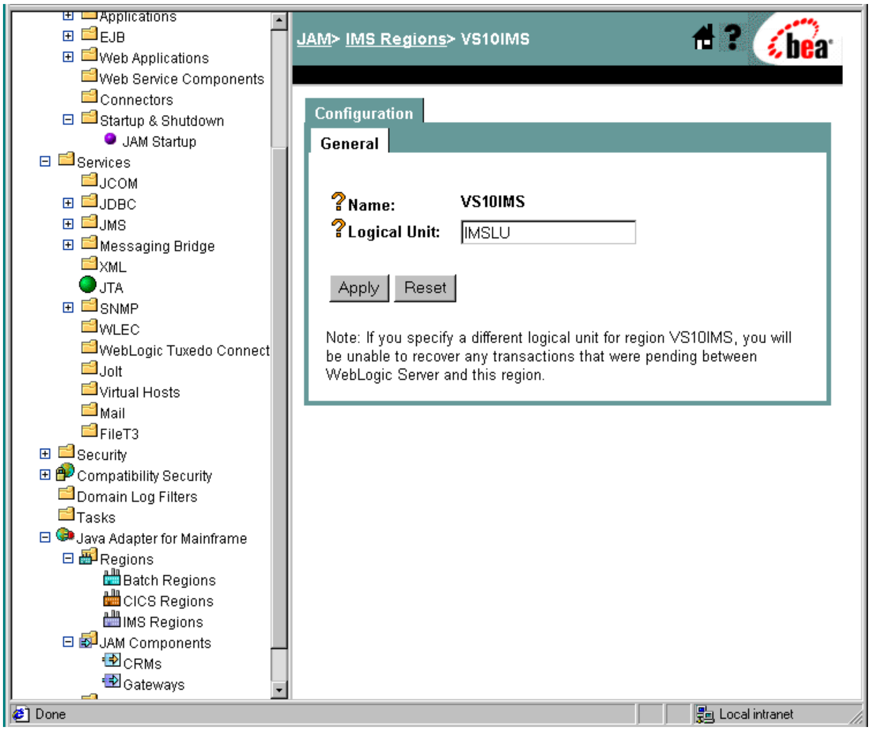
.

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.

Field	Field Description
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 IMS region, click **Java Adapter for Mainframe** → **Regions** → **IMS Regions** in the left pane. In the right pane, click **VS10IMS**. 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. Click **Apply** to set the Logical Unit.

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

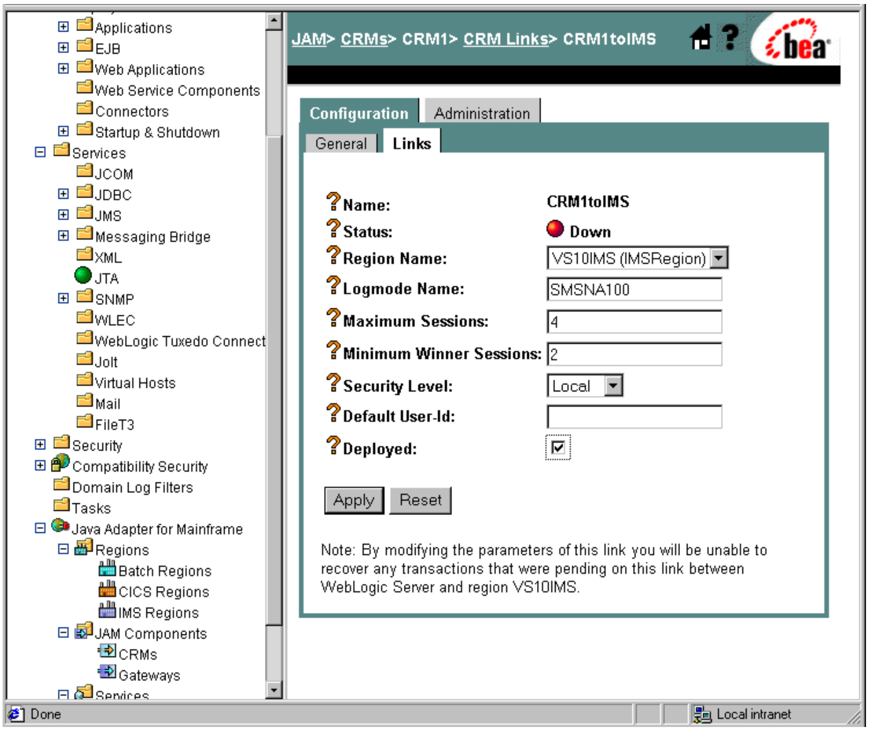


Name: VS10IMS

Logical Unit: IMSLU

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.

c. Click **IMS Regions** at the top of the right pane. In the new window, click **CRM1toIMS**. On the **Links** tab, check **Deployed** and click **Apply**.

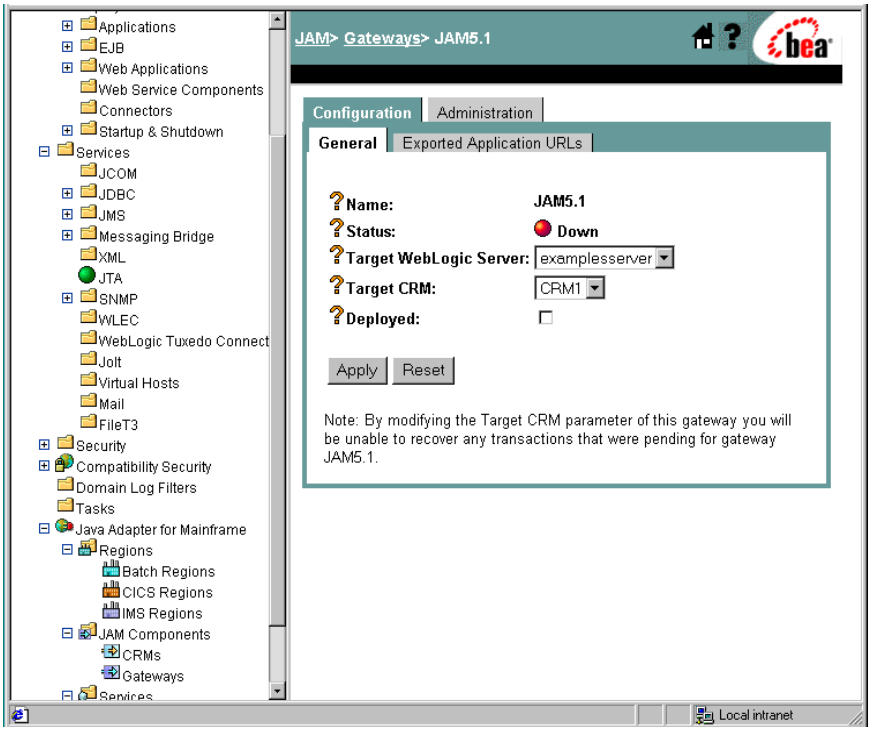


JAM> CRMs> CRM1> CRM Links> CRM1toIMS

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.

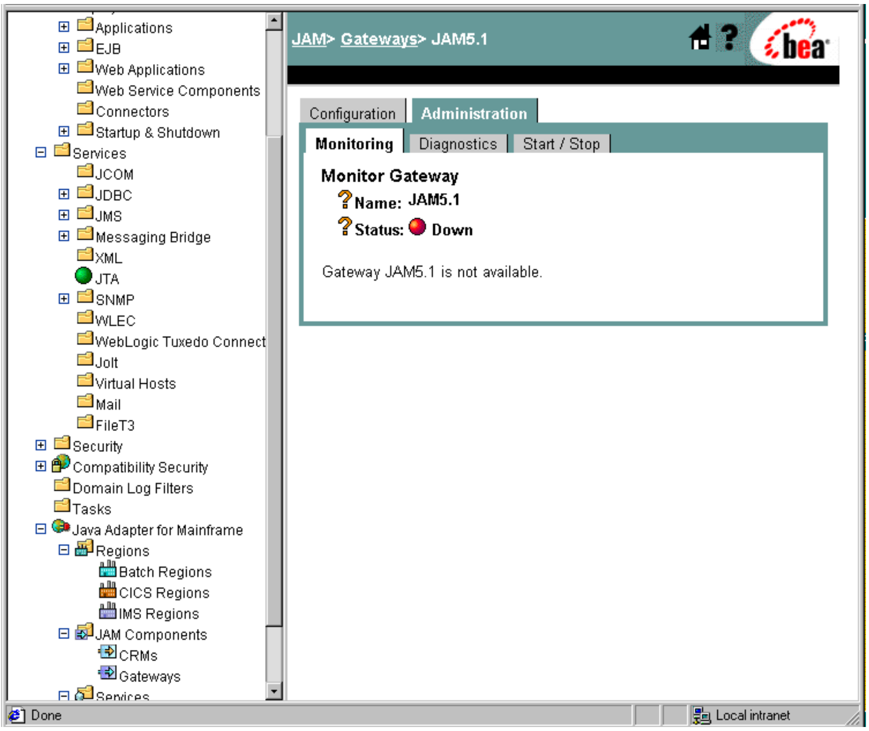
d. In the left pane, click **Gateways**. Click **JAM5.1** in the right pane. On the **General** tab, check **Deployed** and click **Apply**.



Name: JAM5.1
Status: Down
Target WebLogic Server: examplesserver
Target CRM: CRM1
Deployed: ☐

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 Gateway, select the **Administration** tab → **Start/Stop** tab. Click **Start** to start the Gateway.



Monitor Gateway

Name: JAM5.1

Status: Down

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" section, you are ready to set up and run the IMS Sample Application to WebLogic Server JMS topic.

How the Sample Works

This sample illustrates making an asynchronous call through WebLogic 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 WebLogic 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 WebLogic JAM.

The message inserted by `IMSTOJMS` on the IMS message queue is sent to WebLogic 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 WebLogic JAM configuration. The attributes of the `ITOJMSSV` JMSEvent in the WebLogic 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 WebLogic 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`. A directory named `dataviews` is located under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses` directory and the `Chardata.class` is compiled into the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/`

dataviews/examples/IMS/inbound/gateway directory. 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic JAM.

Sample Files

The files for the WebLogic side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/IMS/inbound/gateway
```

The following table lists the sample files and their purpose:

Table 2-1 Files for the WebLogic Application

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.
`build.cmd`	Script that builds the `Chardata` and `TopicReceive` classes. The built class files are under the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\clientclasses\examples\IMS\inbound\gateway` directory.
`build.sh`	Unix script that builds the `Chardata` and `TopicReceive` classes. The built class files are under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/IMS/inbound/gateway` directory.

The files for the IMS side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/IMS/inbound/
mainframe/source
```

The following table lists the sample files and their purpose:

Table 2-2 Files for the IMS Application

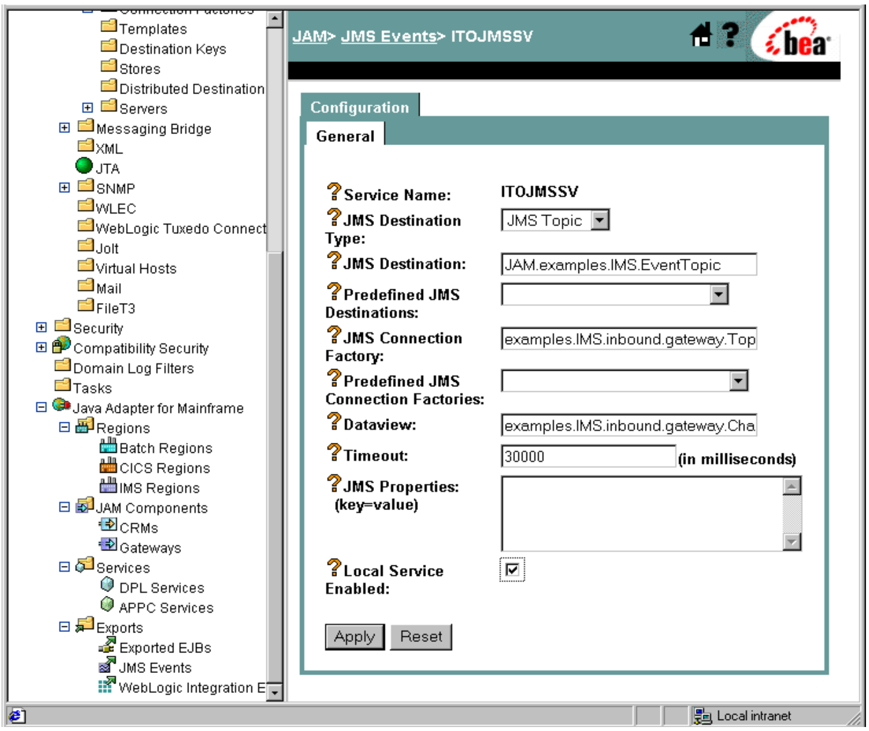
File Name	File Purpose
`COMPIMSC`	JCL that compiles and links the `IMSTOJMS` program.
`IMSINDEF`	Contains sample IMS stage 1 input and `PSBGEN` for the IMS configuration of the `IMSTOJMS` program.
`DFS62DTI`	IMS LU 6.2 descriptor that maps `LTERM` name to Logical Unit and remote transaction name.
`IMSTOJMS`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

Setting Up the Sample

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

Step 1: Enable the Service

To enable the JMS Event, click **Java Adapter for Mainframe** → **Exports**→**JMS Events** in the left pane. Click **ITOJMSSV**. Check **Local Service Enabled** and click **Apply** to enable the Local Service.



Service Name:	ITOJMSSV
JMS Destination Type:	JMS Topic
JMS Destination:	JAM.examples.IMS.EventTopic
Predefined JMS Destinations:	
JMS Connection Factory:	examples.IMS.inbound.gateway.Top
Predefined JMS Connection Factories:	
Dataview:	examples.IMS.inbound.gateway.Cha
Timeout:	30000 (in milliseconds)
JMS Properties: (key=value)	
Local Service Enabled:	✓

Step 2: Set the Environment

Set the environment by executing the `setExamplesEnv` command. From a command prompt, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples` directory and execute the command that corresponds to your system:

For Microsoft Windows:

```
setExamplesEnv.cmd
```

For Unix:

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

The following message will display:

"Your environment has been set."

Step 3: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/IMS/inbound/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the `TopicReceive.class` and the `Chardata.class`. The `TopicReceive.class` is put in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/clientclasses/examples/IMS/inbound/gateway` directory. The `Chardata.class` file is put in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/clientclasses/dataview/examples/IMS/inbound/gateway` directory.
- 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.

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

Step 4: Run the TopicReceive Program

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"
```

In this statement, the following definitions apply:

`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”
```

Step 5: Complete Mainframe Tasks

On the machine with the IMS 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 from the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/IMS/inbound/mainframe/source` directory to the PDS that you created:
   - `IMSTOJMS`
   - `IMSINDEF`
   - `DFS62DTI`
   - `COMPIMSC`
3. In the `COMPIMSC` JCL, make the following changes:
   - Change the `JOB` statement.
   - Change `YOURHLQ.SOURCE` to the PDS you created.
   - Change `YOUR.PGMLIB` to a `PGMLIB` for your IMS region.

- Change `YOUR.PROCLIB` to the location of `CBLTDLI` (usually `IMS.PROCLIB`).
- Change `YOUR.RESLIB` to the proper value.
- 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.

4. Submit the `COMPIMSC` JCL. Make sure that all condition codes are 0.
5. Define the program `IMSTOJMS` to the IMS region. `IMSINDEF` contains sample IMS stage 1 and `PSBGEN` input. See your IMS systems programmer for assistance.
6. Define the `LTERM`. `DFS62DTI` contains sample definition for an APPC `LTERM`. See your IMS systems programmer for assistance.
   - Change the Logical Unit name to match the Logical Unit configured for the CRM in Step 4 of the "Preparing to Use the IMS Samples" section.
   - If you change the `LTERM` name, make sure to change it at the command prompt when you run the sample. See the "Running the Sample" section.
   - Do not change the `TPNAME`. It is configured to match the name of the `JMSEVENT` in the WebLogic JAM configuration.

Running the Sample

To run the sample, complete the following steps:

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

   `IMSTOJMS JAMIMS01 hello` (or any other string)

   The string will be displayed 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 the "Step 5: Complete Mainframe Tasks" section, you must enter that `LTERM` name at the command prompt instead of `JAMIMS01`.

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 the "Preparing to Use the IMS Samples" section, you are ready to set up and run the Java client to IMS sample application.

Understanding How the Sample Works

This sample demonstrates requests from a Java client through WebLogic 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 WebLogic 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 WebLogic 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.

Sample Files

The files for the WebLogic side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/IMS/outbound/
gateway
```

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

Table 2-3 Files for the WebLogic Application

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 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.
`build.cmd`	Script that builds the `IvtnoInRecord`, `IvtnoOutRecord`, `BaseClient` and `Client` classes. The built class files are under the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\clientclasses\examples\IMS\outbound\gateway` directory.
`build.sh`	Unix script that builds the `IvtnoInRecord`, `IvtnoOutRecord`, `BaseClient` and `Client` classes. The built class files are under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/IMS/outbound/gateway` directory.

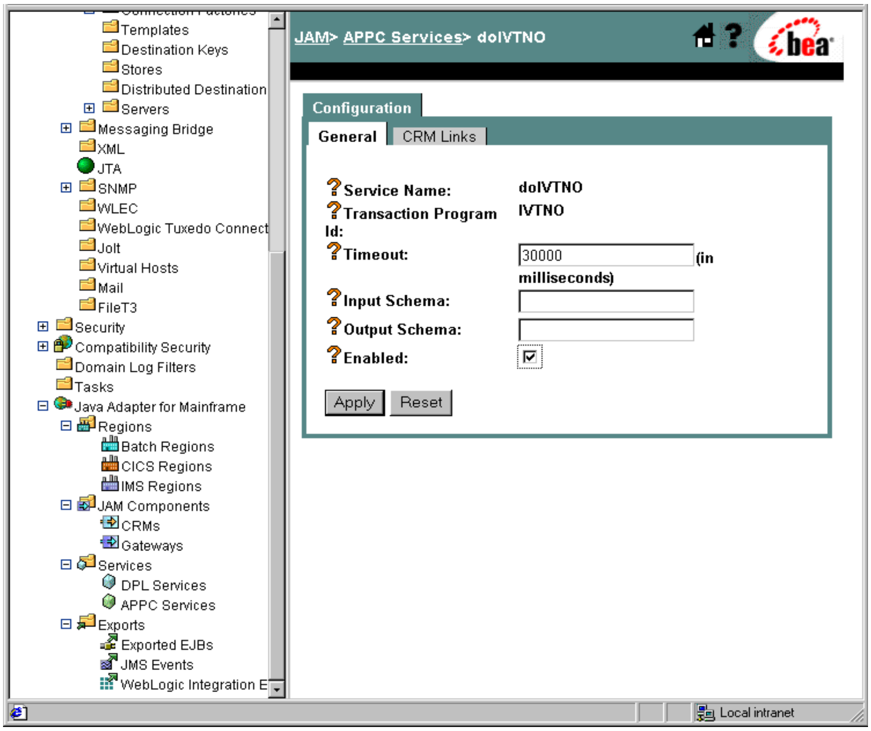
Setting Up the Sample

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.

Step 1: Enable the Service

To enable the APPC service, click **Java Adapter for Mainframe →Services→APPC Services** in the left pane. In the right pane, click **doIVTNO** under **Service Name**. Check **Enabled** and click **Apply**.



Service Name: doIVTNO
Transaction Program Id: IVTNO
Timeout: 30000 (in milliseconds)
Input Schema:
Output Schema:
Enabled: ☑

Step 2: Set the Environment

On the machine from which the sample client is run, set the environment by performing the following step.

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

- 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: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/IMS/outbound/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the client classes and put them in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/IMS/outbound/gateway` directory.
- 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 *BEA WebLogic 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 WebLogic 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 the "Preparing to Use the IMS Samples" section, you are ready to set up 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 WebLogic 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 WebLogic 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 WebLogic 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.

Sample Files

The files for the WebLogic side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/transactional/IMS/
outbound/gateway
```

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

Table 2-4 Files for the WebLogic Application

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.
`build.cmd`	Script that builds the `IvtnoInRecord`, `IvtnoOutRecord`, `BaseClient` and `Client` classes. The built class files are under the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\clientclasses\examples\transactional\IMS\outbound\gateway` directory.

File Name	File Purpose
`build.sh`	Unix script that builds the `IvtnoInRecord`, `IvtnoOutRecord`, `BaseClient` and `Client` classes. The built class files are under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/transactional/IMS/outbound/gateway` directory.

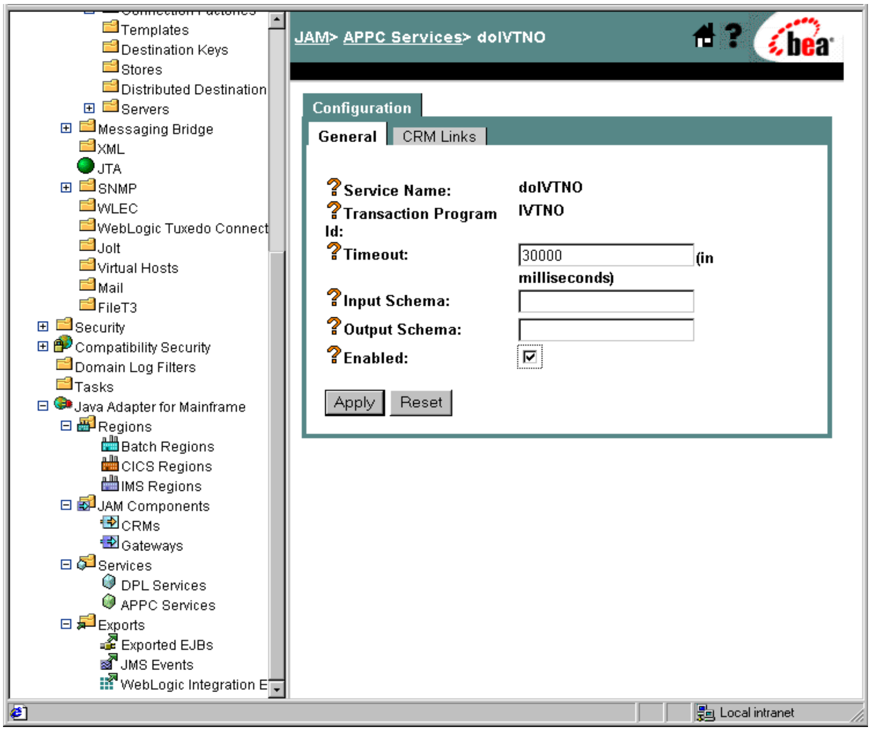
Setting Up Sample

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.

Step 1: Enable the Service

To enable the APPC service, click **Java Adapter for Mainframe** → **Services**→**APPC Services** in the left pane. In the right pane, click **doIVTNO**. Check **Enabled** and click **Apply**.



Service Name: doIVTNO
Transaction Program Id: IVTNO
Timeout: 30000 (in milliseconds)
Input Schema:
Output Schema:
Enabled: ✓

Step 2: Set the Environment

On the machine from which the sample client is to be run, set the environment by performing the following step.

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

- From a command prompt, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples` directory 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: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/transactional/IMS/outbound/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the client classes and put them in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/transactional/IMS/outbound/gateway` directory.
- 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 *BEA WebLogic Java Adapter for Mainframe Programming Guide*.

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 WebLogic 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
```

3 Using the CICS Samples

The CICS samples demonstrate how BEA WebLogic Java Adapter for Mainframe (WebLogic JAM) integrates the WebLogic applications with CICS applications. This section provides the following information:

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

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 WebLogic 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 purchase. 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 the "Before You Run the Samples" section.

2. Prepare to use the CICS sample.

   a. Start the CRM.

   b. Update the WebLogic JAM configuration file.

   c. Start the `examples` domain.

   d. Configure the WebLogic 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 the "Before You Run the Samples" section, you are ready to use the sample. Information about how to use the CICS samples is presented in the following sections:

- Preparing to Use the CICS Samples
- Using the CICS Application to WebLogic Server Sample EJB
  - Understanding How the Sample Works
  - Setting up the Sample
  - Running the Sample
- Using the Java Client to CICS Sample Application
  - Understanding How the Sample Works
  - Setting up the Sample
  - Running the Sample
- Using the Transactional Sample from WebLogic Server to CICS
  - Understanding How the Sample Works
  - Setting up the Sample
  - Running 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 WebLogic 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.

Starting the CRM on z/OS or OS/390 Unix

On z/OS or OS/390 Unix, you may use a script to start the CRM. Scripts are installed with the Gateway in the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/crm/unix` directory. The script, `crm.env`, appends the necessary values to your environment variables. The script, `startcrm.sh`, starts the CRM. To use these scripts, complete the following steps:

1. FTP the following two scripts to the directory from which the CRM will run:
   - `crm.env`
   - `startcrm.sh`
2. Edit `crm.env`. Supply the correct values for the `APPDIR` and `CRMDIR` variables. `APPDIR` is the directory from which the CRM will run. `CRMDIR` is the CRM installation directory.
3. Edit `startcrm.sh`. To use a different port than the default port, `7101`, change the port number. However, if you change the port number, make sure to change it in the corresponding field in the WebLogic Administration Console CRM1 pane. You do not need to change the address because the script will run on the machine where the CRM is installed.

Note: BEA recommends that you do not change the CRM name from CRM1. This name for the CRM is preconfigured for all of the samples.

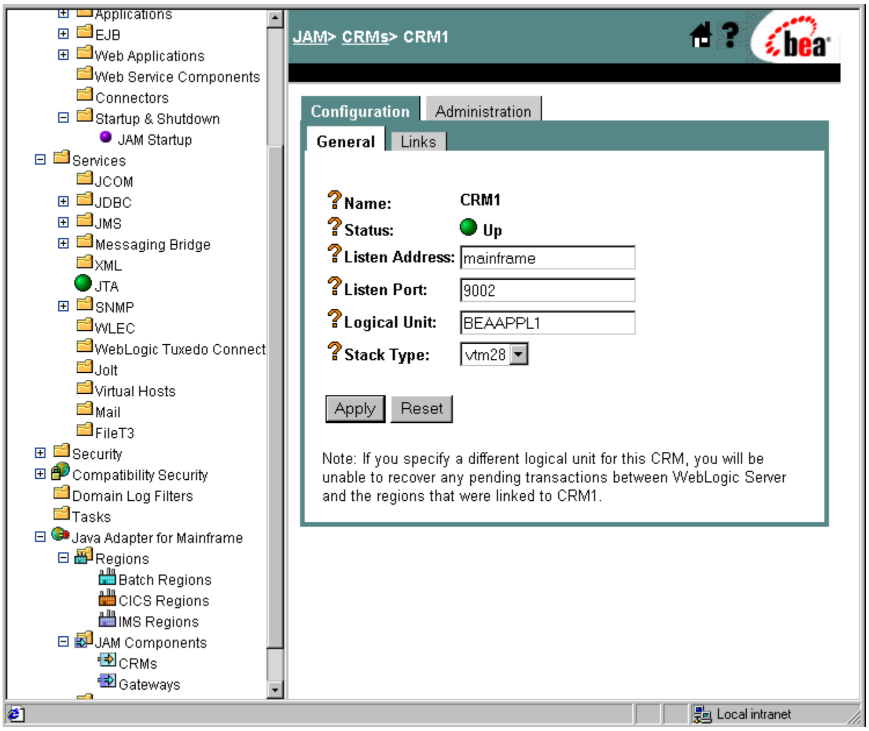
4. Execute the `startcrm.sh` script:

```
. ./startcrm.sh
```

Compare Figure 3-1 with the script in Listing 3-1. Notice how the parameters in the script correspond to the fields in the WebLogic Administration Console. The script illustrates the values for `startcrm.sh` script parameters for running the samples.

Note: The port number is `7101`. You can change the port number; however, if the port number is changed make sure to change it in the corresponding field in the Gateway configuration CRM1 pane of the WebLogic Administration Console.

Figure 3-1 Fields for the CRM



Name: CRM1
Status: Up
Listen Address: mainframe
Listen Port: 9002
Logical Unit: BEAAPPL1
Stack Type: vtm28

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.

Listing 3-1 Command to Run the CRM

```
$CRMDIR/bin/CRM //127.0.0.1:7101 CRM1 < /Dev/null > std.out
2>std.err &
```

Starting the CRM on z/OS or OS/390 MVS

On z/OS or OS/390 MVS, start the CRM by submitting the `CRMSTART` JCL that is installed with the CRM. The `CRMSTART` JCL must be modified for your environment. For information about modifying the `CRMSTART` JCL, see the *BEA WebLogic Java Adapter for Mainframe Configuration and Administration Guide*.

As you modify the `CRMSTART` JCL, make sure that you note the following parameters in the value of the `STARTCMD` parameter in the JCL. These parameters correspond to fields in the WebLogic Administration Console. These values must be the same in the JCL and in the WebLogic Administration Console.

- The machine address where the CRM will run
- The port number on which the CRM will listen
- The name by which the Gateway will refer to the CRM

Compare Figure 3-1 with the JCL in Listing 3-2. Notice how the parameters in the JCL correspond to the fields in the WebLogic Administration Console. The JCL illustrates the values for `STARTCMD` parameters for running the samples.

- The machine where the CRM will run is `myhost` in this sample. You must replace `myhost` with the hostname or IP address of your mainframe to allow IP-based communication from the Windows or Unix machine where WebLogic Server is running. You may verify this parameter with the `ping` command on Windows or Unix.
- The port number is `7101`. You can change the port number; however, if you change the port number, make sure to change it in the corresponding field in the Gateway configuration CRM1 pane of the WebLogic Administration Console.

Note: BEA recommends that you do not change the CRM name from CRM1, because this name for the CRM is preconfigured for all of the samples.

Listing 3-2 The `STARTCMD` parameter in the `CRMSTART` JCL

```
//  SET STARTCMD='"//myhost:7101" CRM1'
```

Step 2: Update the WebLogic JAM Configuration File

On the machine where the Gateway is located, update the WebLogic JAM configuration file from the command prompt by completing the following steps:

1. Locate the `jamconfig_CICS.xml` file under the following directory:

   `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples`

2. Copy `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:

- For Microsoft Windows:

  `startExamplesServer.cmd`

- For Unix:

  `. ./startExamplesServer.sh`

Step 4: Configure the WebLogic JAM Gateway

Most configuration tasks are preconfigured or were completed during the installation process by the installer program. For additional information about configuring WebLogic JAM, see the *BEA WebLogic 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:

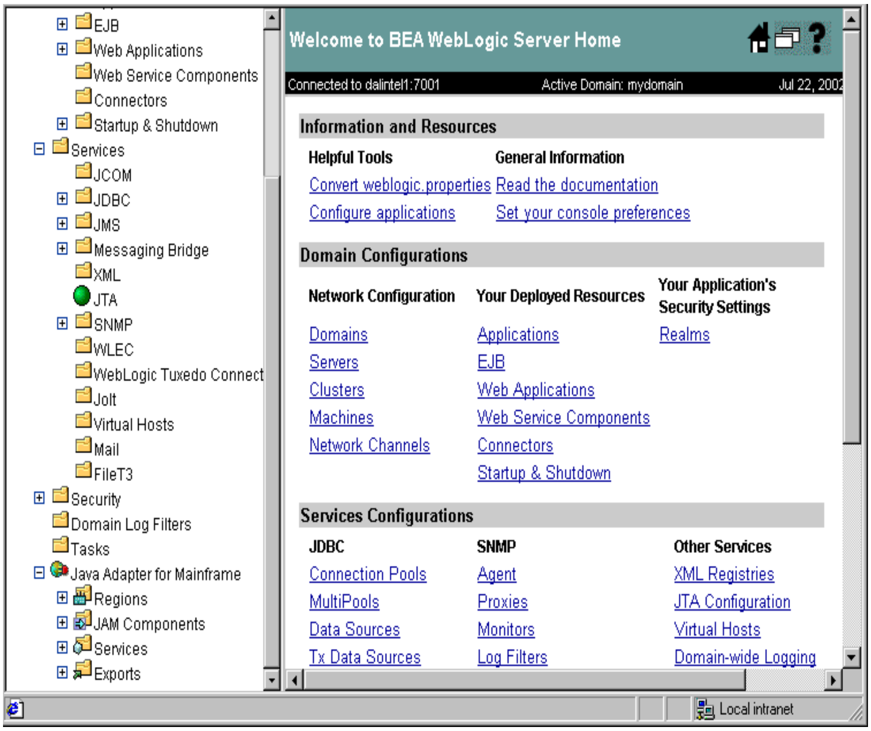
   - user: system

     This user name cannot be changed.

   - password: security

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

The WebLogic Administration Console displays.



Welcome to BEA WebLogic Server Home

Connected to dalintel1:7001 Active Domain: mydomain Jul 22, 2002

Information and Resources

Helpful Tools	General Information
Convert weblogic.properties	Read the documentation
Configure applications	Set your console preferences

Domain Configurations

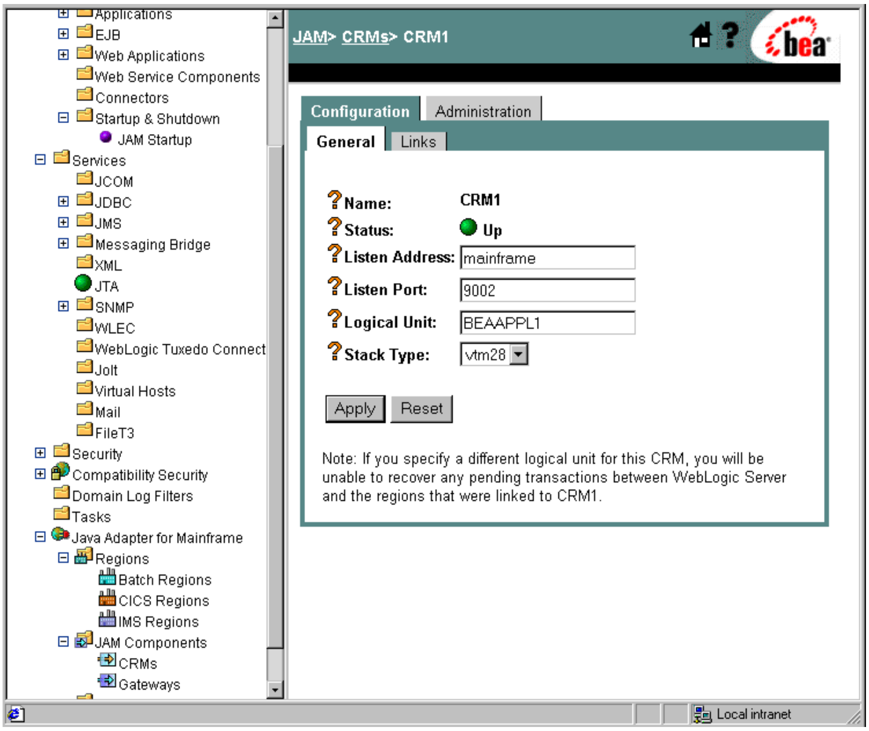
Network Configuration	Your Deployed Resources	Your Application's Security Settings
Domains	Applications	Realms
Servers	EJB	
Clusters	Web Applications	
Machines	Web Service Components	
Network Channels	Connectors	
	Startup & Shutdown	

Services Configurations

JDBC	SNMP	Other Services
Connection Pools	Agent	XML Registries
MultiPools	Proxies	JTA Configuration
Data Sources	Monitors	Virtual Hosts
Tx Data Sources	Log Filters	Domain-wide Logging

3. To configure the CRM to the WebLogic 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.



Name: CRM1
Status: Up
Listen Address: mainframe
Listen Port: 9002
Logical Unit: BEAAPPL1
Stack Type: vtm28

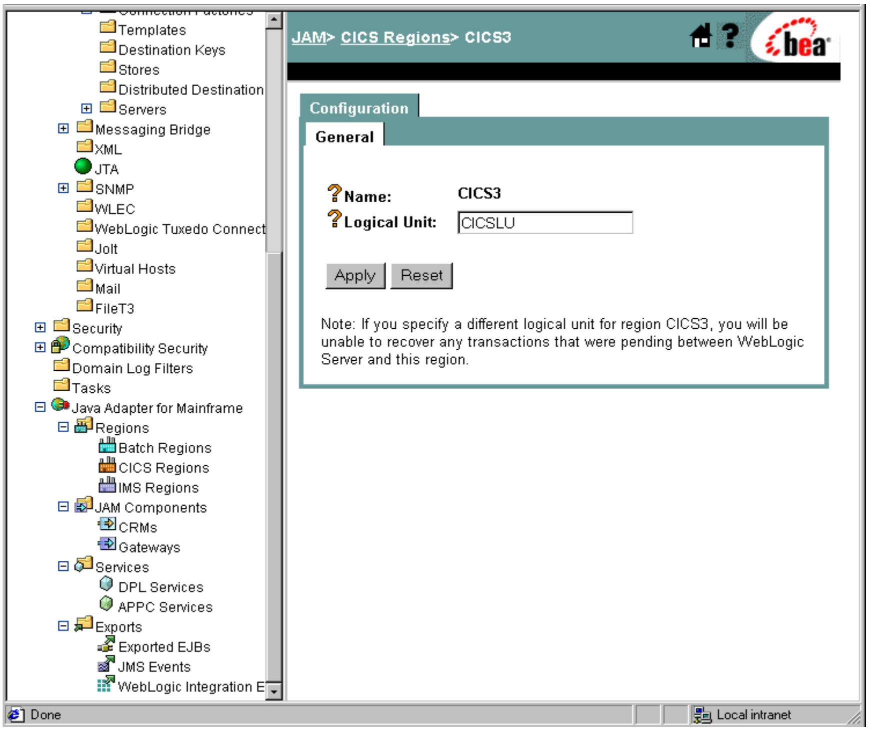
Note: If you specify a different logical unit for this CRM, you will be unable to recover any pending transactions between WebLogic Server and the regions that were linked to CRM1.

.

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).

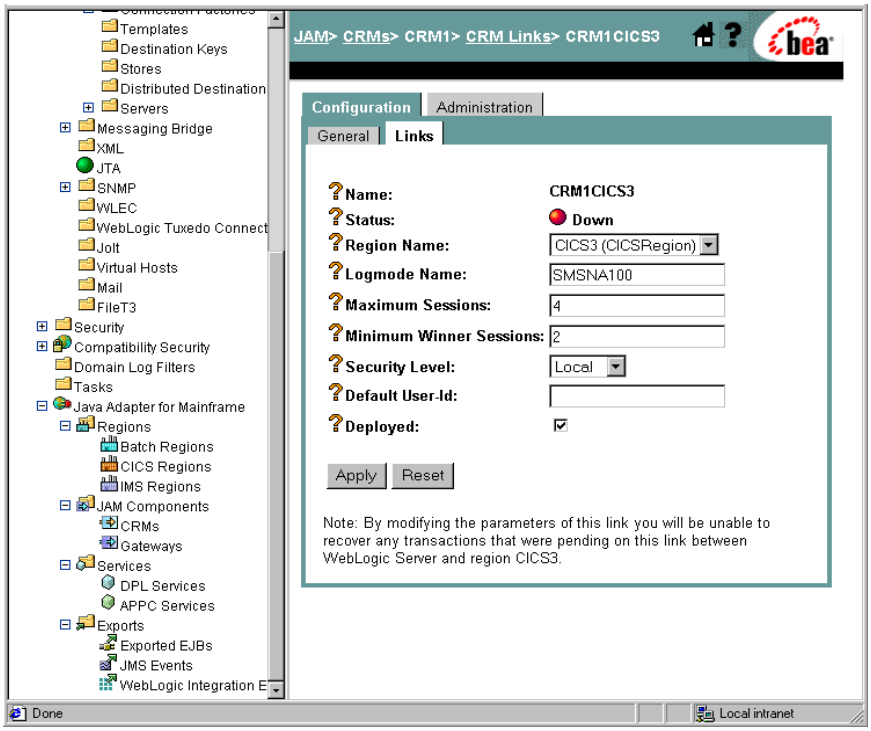


Name: CICS3

Logical Unit: CICSLU

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.

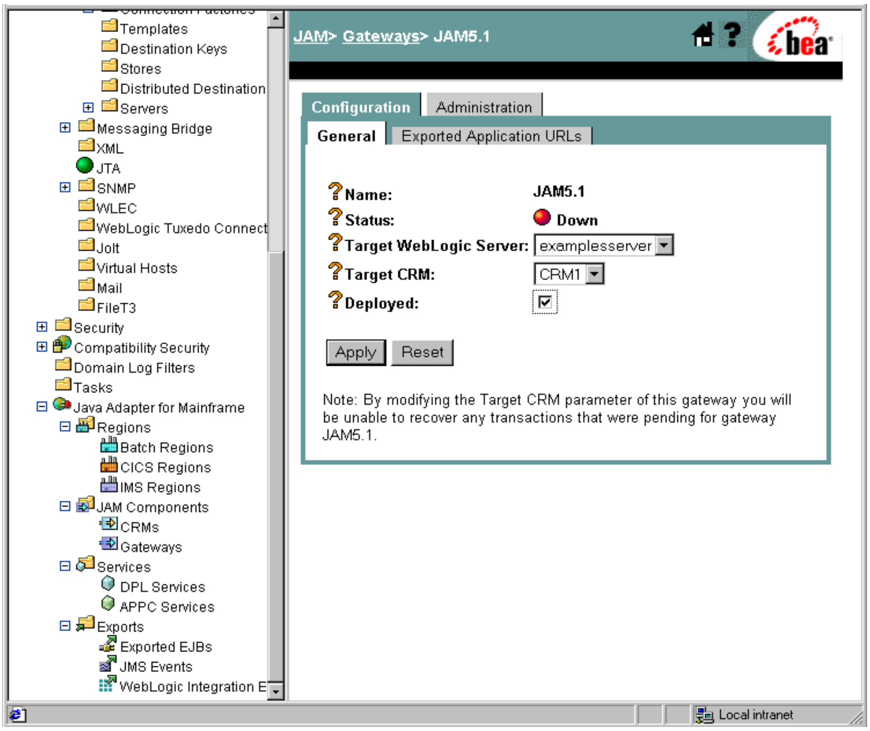
c. Click **CICS Regions** at the top of the pane. Click **CRM1CICS3**. In the new window, click **CRM1CICS3**. On the **Links** tab, check **Deployed** and click **Apply**.



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.

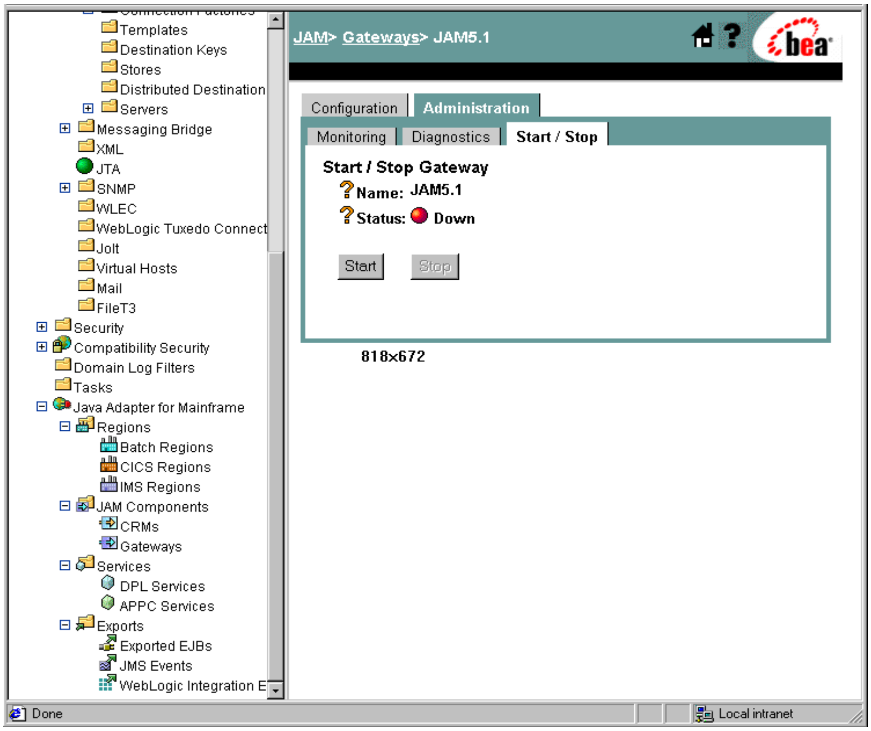
d. In the left pane, click **JAM Components→Gateways**. Click **JAM5.1** in the right pane. On the **General** tab, check **Deployed**. Click **Apply**.



Name: JAM5.1
Status: Down
Target WebLogic Server: examplesserver
Target CRM: CRM1
Deployed: ☑

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



Start / Stop Gateway

Name: JAM5.1

Status: Down

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 the "Preparing to Use the CICS Samples" 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic JAM to a service offered by an EJB deployed in WebLogic Server.

Sample Files

The files for the WebLogic JAM side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/inbound/
gateway
```

The following table lists the sample files and their purpose:

Table 3-1 Files for the WebLogic Application

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.

File Name	File Purpose
`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.
`build.cmd`	Script that builds the `TradeRecord`, `TradeServer`, `TradeServerBean`, `TradeServerHome`, and `ExtTradeServerBean` classes. It assembles the classes along with the necessary deployment descriptors into a `.jar` file. The resulting `JAM_TradeServer.jar` file is under the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\applications\examples\CICS\inbound\gateway` directory.
`build.sh`	Unix script that builds the `TradeRecord`, `TradeServer`, `TradeServerBean`, `TradeServerHome`, and `ExtTradeServerBean` classes. It assembles the classes along with the necessary deployment descriptors into a `.jar` file. The resulting `JAM_TradeServer.jar` file is under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/applications/examples/CICS/inbound/gateway` directory.
`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 files for CICS side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/inbound/
mainframe/source
```

The following table lists the sample files and their purpose:

Table 3-2 Files for CICS Application

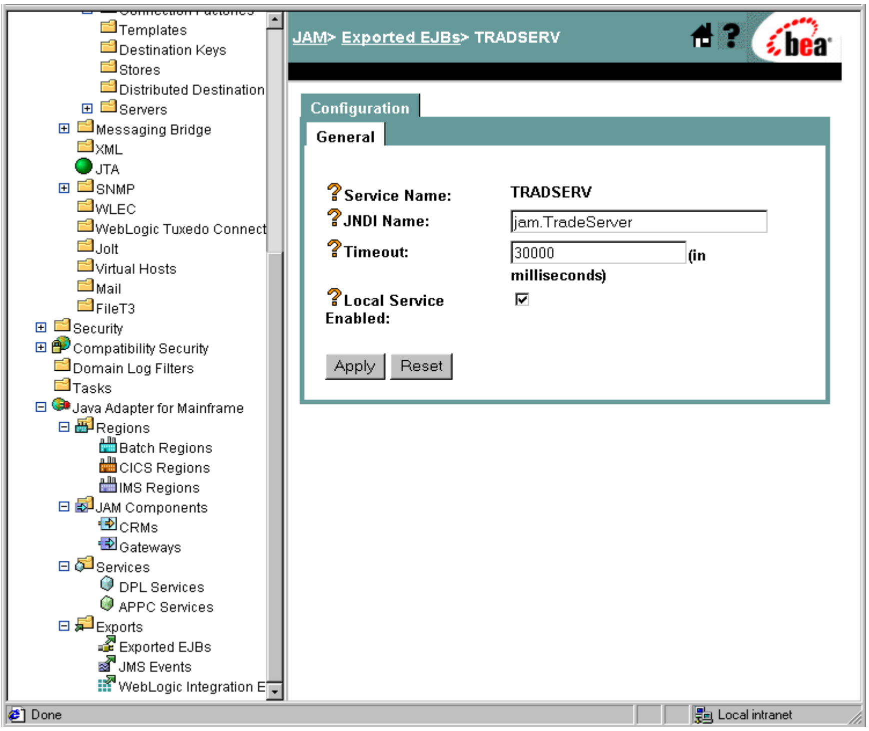
File Name	**File Purpose**
`TRADRCRD`	COBOL copybook that defines the structure of the mainframe data.
`CMPPROC`	Procedure used to compile and link the CICS programs.
`COMPTRCL`	JCL that executes the `CMPROC` for the program `TRADCLNT`.
`CSDUPDTR`	RDO cards to define `TRADCLNT` and the service `TRADSERV` 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 WebLogic 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.



Service Name: TRADSERV
JNDI Name: jam.TradeServer
Timeout: 30000 (in milliseconds)
Local Service Enabled: ✓

Step 2: Set the Environment

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

- 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: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/inbound/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the bean classes and put them in the `JAM_TradeServer.jar` at `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/applications`.
- Run `egencobol` to use the eGen Application Code Generator on `tradeserver.egen` to generate the code for the `TradeServer` bean and it deployment descriptors.

  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 *BEA WebLogic Java Adapter for Mainframe Programming Guide*.

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 from the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/inbound/mainframe/source` directory into the PDS that you created:
   - `CMPPROC`
   - `COMPTRCL`
   - `CSDUPDTR`
   - `CSDUTRCL`
   - `TRADCLNT`
   - `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:
   - Change the `JOB` statement.
   - Change the `JCLLIB ORDER` and the `PDSSRC` to the PDS you created.
   - 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:
   - Change the name of the `LIST` to a valid list name for your CICS region.
   - Change the value of `REMOTESYSTEM` to the connection defined for the CRM.
   - If you change the `GROUP` name, make sure you change it everywhere.

7. In the `CSDUPDTR` JCL, make the necessary changes to these statements:
   - `JOB`
   - `STEPLIB`
   - `DFHCSD`
   - `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 WebLogic JAM sample.

    To install the WebLogic 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 the "Preparing to Use the CICS Samples" section, 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic JAM.

Sample Files

The files for the WebLogic JAM side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/outbound/
gateway
```

The following table lists the sample files and their purpose:

Table 3-3 Files for WebLogic JAM Application

File Name	File Purpose
`emprec.cpy`	COBOL copybook that defines the structure of the mainframe employee record.
`emprec.egen`	eGen script that generates the `EmployeeRecord.java` DataView class.
`EmployeeRecord.java`	DataView class that corresponds to the `emprec.cpy` COBOL copybook.
`baseClient.egen`	eGen script that generates the `EmployeeRecord.java` DataView class and the `BaseClient.java EgenClient` class.
`BaseClient.java`	Java class that extends `EgenClient` class that calls the various mainframe services.
`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 mainframe service calls to the user. It invokes the mainframe services by calling the `callService` method of its `BaseClient` member.
`build.cmd`	Script that builds the `EmployeeRecord`, `BaseClient`, and `Client` classes. The built class files are under the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\clientclasses\examples\CICS\outbound\gateway` directory.
`build.sh`	Unix script that builds the `EmployeeRecord`, `BaseClient`, and `Client` classes. The built class files are under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/CICS/outbound/gateway` directory.

The files for CICS side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/outbound/mainf
rame/source
```

The following table lists the sample files and their purpose:

Table 3-4 Files for CICS Application

File Name	File Purpose
`EMPREC`	COBOL copybook that defines the structure of the employee record data.
`DPLDEMOC`	CICS server program that imitates the insertion of a record on a database by creating a TS queue.
`DPLDEMOR`	CICS server program that imitates the reading of a record on a database by reading a TS queue.
`DPLDEMOU`	CICS server program that imitates the update of a record on a database by updating a TS queue.
`DPLDEMOD`	CICS server program that imitates the deletion of a record on a database by deleting a TS queue.
`CMPPROC`	Procedure used to compile and link the CICS programs.
`COMPCRUD`	JCL that executes the `CMPROC` for the programs `DPLDEMOC`, `DPLDEMOR`, `DPLDEMOU`, and `DPLDEMOD`.
`CSDUCRUD`	RDO cards to define `DPLDEMOC`, `DPLDEMOR`, `DPLDEMOU`, and `DPLDEMOD` to the CICS region.
`CSDUPDCO`	JCL that executes `CSDUCRUD`.

Setting Up the Sample

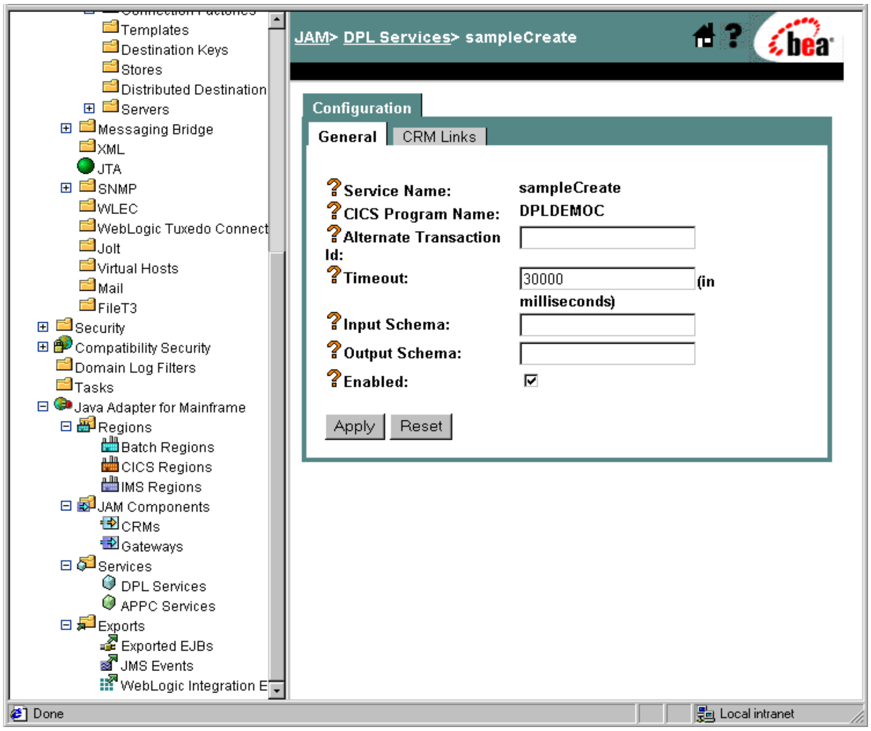
To set up the Java client to CICS sample application, complete the following steps.

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`



Service Name: sampleCreate
CICS Program Name: DPLDEMOC
Alternate Transaction Id:
Timeout: 30000 (in milliseconds)
Input Schema:
Output Schema:
Enabled: ☑

Step 2: Set the Environment

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

- From a command prompt, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples` directory and 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: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/outbound/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the client classes and put them in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/CICS/outbound/gateway` directory.
- Run `egencobol` to use the eGen Application Code Generator on:
  - `emprec.egen` to generate `EmployeeRecord.java`
  - `baseClient.egen` to generate `BaseClient.java` and `EmployeeRecord.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 *BEA WebLogic Java Adapter for Mainframe Programming Guide*.

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 from the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/CICS/outbound/mainframe/source` directory 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:
   - Change the `JOB` statement.
   - Change the `JCLLIB ORDER` and the source to the PDS you created.
   - 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:
   - Change the name of the `LIST` to a valid list name for your CICS region.
   - You can also change the `GROUP` name, but make sure you change it everywhere.

7. In the `CSDUPDCO` JCL, make the following changes:
   - Change the `JOB` statement.
   - Change the `STEPLIB`.
   - Change the `DFHCSD`.
   - 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 WebLogic JAM sample.

    To install the WebLogic 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:

- The first pair of command line switches are optional and represent the `hostname` and `port` for a remote machine on which the WebLogic 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.

- The second pair of command line switches are mandatory.
  - `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.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 the "Preparing to Use the CICS Samples" section, 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 WebLogic 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`, `DPLDEMVU`, 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 WebLogic 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 WebLogic 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`
- `DPLDEMVU`
- `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 WebLogic JAM.

Sample Files

The files for the WebLogic JAM side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/transactional/CICS/
outbound/gateway
```

The following table lists the sample files and their purpose:

Table 3-5 Files for WebLogic JAM Application

File Name	File Purpose
`emprec.cpy`	COBOL copybook that defines the structure of the mainframe employee record.
`emprec.egen`	eGen script that generates the `EmployeeRecord.java` DataView class.
`EmployeeRecord.java`	DataView class that corresponds to the `emprec.cpy` COBOL copybook.

File Name	File Purpose
`baseClient.egen`	eGen script that generates the `EmployeeRecord.java` DataView class and the `BaseClient.java EgenClient` class.
`BaseClient.java`	Java class that extends `EgenClient` class that calls the various mainframe services.
`Client.java`	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 `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 checking the contents of the JMS queue.
`build.cmd`	Script that builds the `EmployeeRecord`, `BaseClient`, and `Client` classes. The built class files are under the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\clientclasses\examples\transactional\CICS\outbound\gateway` directory.
`build.sh`	Unix script that builds the `EmployeeRecord`, `BaseClient`, and `Client` classes. The built class files are under the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/examples/transactional/CICS/outbound/gateway` directory.

The files for CICS side of the sample are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/transactional/CICS/
outbound/mainframe/source
```

The following table lists the samples files and their purpose:

Table 3-6 Files for CICS Application

File Name	File Purpose
`EMPREC`	COBOL copybook that defines the structure of the employee record data.
`DPLDEMVC`	CICS server program that inserts a record in a VSAM file.

File Name	File Purpose
`DPLDEMVR`	CICS server program that reads a record in a VSAM file.
`DPLDEMVU`	CICS server program that updates a record in a VSAM file.
`DPLDEMVD`	CICS server program that deletes a record in a VSAM file.
`CMPPROC`	Procedure used to compile and link the CICS programs.
`COMPILEV`	JCL that executes the `CMPROC` for the programs `DPLDEMVC`, `DPLDEMVR`, `DPLDEMVU`, and `DPLDEMVD`.
`JVSAMRDO`	RDO cards to define `DPLDEMVC`, `DPLDEMVR`, `DPLDEMVU`, and `DPLDEMVD` programs and the `BEAJAMTV` VSAM file to the CICS region.
`CSDUPDCT`	JCL that executes `JVSAMRDO`.
`JAMVSAMC`	Delete/Define cards for the VSAM file.
`BLDVSAM`	JCL that invokes `IDCAMS` for `JAMVSAMC`.

Setting Up the Sample

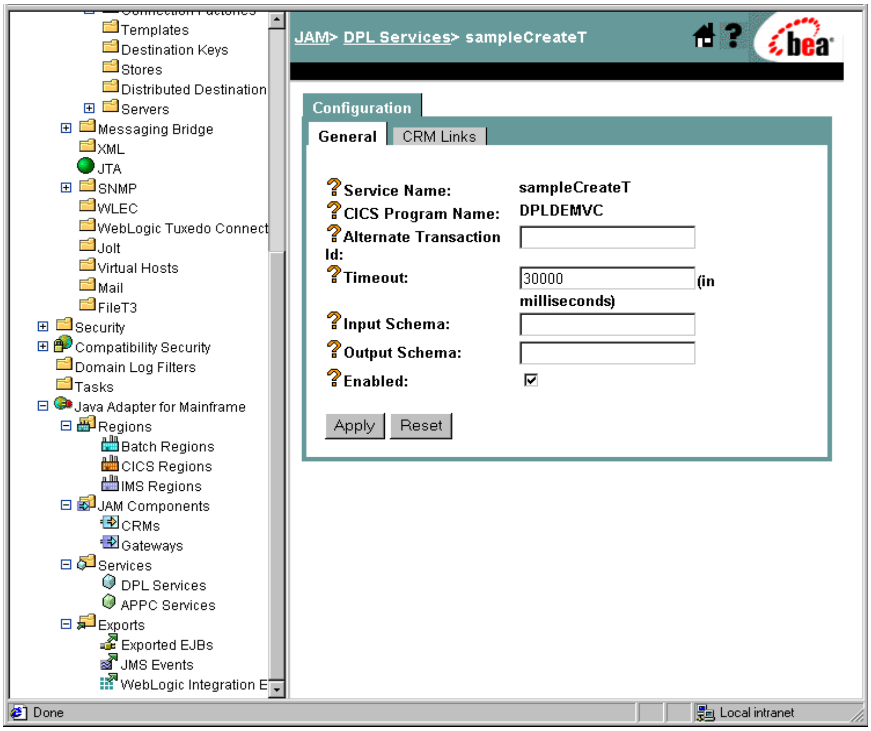
To set up the transactional sample from WebLogic Server to CICS, complete the following steps.

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`



Service Name:	sampleCreateT
CICS Program Name:	DPLDEMVC
Alternate Transaction Id:	
Timeout:	30000 (in milliseconds)
Input Schema:	
Output Schema:	
Enabled:	✓

Step 2: Set the Environment

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

- From a command prompt, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples` directory 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: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/transactional/CICS/outbound/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the client classes and put them in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/clientclasses/transactional/CICS/outbound/gateway` directory.
- Run `egencobol` to use the eGen Application Code Generator on:
  - `emprec.egen` to generate `EmployeeRecord.java`
  - `baseClient.egen` to generate `BaseClient.java` and `EmployeeRecord.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 *BEA WebLogic Java Adapter for Mainframe Programming Guide*.

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 from the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/transactional/CICS/outbound/mainframe/source` directory into the PDS that you created:
   - `EMPREC`
   - `DPLDEMVC`
   - `DPLDEMVR`
   - `DPLDEMVU`
   - `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:
   - Change the `JOB` statement.
   - Change the `JCLLIB ORDER` and the `SOURCE` to the PDS you created.
   - 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:
   - Change all occurrences of YOURHLQ.JAM.VSAMFILE to the name you choose for the VSAM file used in this sample.
   - Change YOURVOL to an appropriate volume ID.
7. In the BLDVSAM JCL, make the following changes:
   - Change the JOB statement.
   - 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:
   - Change the name of the LIST to a valid list name for your CICS region.
   - You can also change the GROUP name, but make sure it is changed everywhere.
   - 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:
    - Change the JOB statement.
    - Change the STEPLIB.
    - Change the DFHCSD.
    - 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 WebLogic JAM sample.

    To install the WebLogic 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`, `DPLDEMVU`, 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:

- The first pair of command line switches are optional, and represent the `hostname` and `port` for a remote machine on which the WebLogic 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, 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 Sample

The explicit APPC sample demonstrates how BEA WebLogic Java Adapter for Mainframe (WebLogic JAM) integrates the WebLogic applications with batch MVS COBOL applications. This section provides the following information:

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

About the Explicit APPC Sample

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 the "Using the Sample" section.

Batch MVS COBOL Client to WebLogic EJB Sample

This sample demonstrates the functional capability of BEA WebLogic Java Adapter for Mainframe (WebLogic 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.

Roadmap for the Sample

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

1. Verify prerequisite tasks.

   For a listing of prerequisite tasks, see the "Before You Run the Samples" section.

2. Prepare to use the explicit APPC sample.

   a. Start the CRM.

   b. Set Logical Unit VTAM definitions.

   c. Update the WebLogic JAM Configuration File

   d. Start the `examples` domain

   e. Configure the WebLogic 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 Sample

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 WebLogic 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.

Starting the CRM on z/OS or OS/390 Unix

On z/OS or OS/390 Unix, you may use a script to start the CRM. Scripts are installed with the Gateway in the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/crm/unix` directory. The script, `crm.env`, appends the necessary values to your environment variables. The script, `startcrm.sh`, starts the CRM. To use these scripts, complete the following steps:

1. FTP the following two scripts to the directory from which the CRM will run:
   - `crm.env`
   - `startcrm.sh`
2. Edit `crm.env`. Supply the correct values for the `APPDIR` and `CRMDIR` variables. `APPDIR` is the directory from which the CRM will run. `CRMDIR` is the CRM installation directory.
3. Edit `startcrm.sh`. To use a different port than the default port, `7101`, change the port number. However, if you change the port number, make sure to change it in the corresponding field in the WebLogic Administration Console CRM1 pane. You do not need to change the address because the script will run on the machine where the CRM is installed.

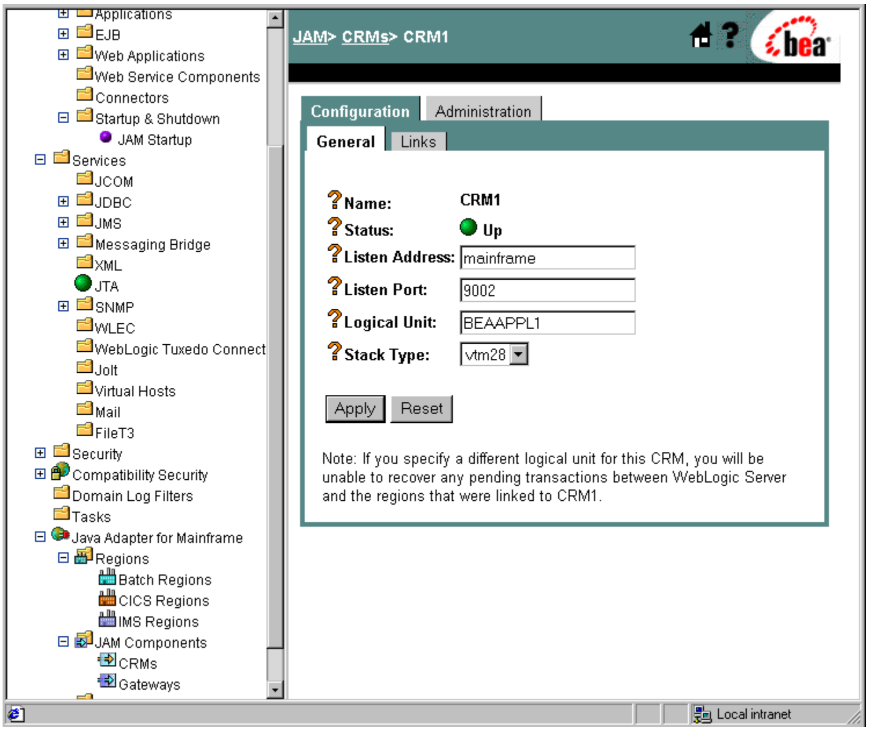
   Note: BEA recommends that you do not change the CRM name from CRM1. This name for the CRM is preconfigured for all of the samples.
4. Execute the `startcrm.sh` script:

   ```
   . ./startcrm.sh
   ```

Compare Figure 4-1 with the script in Listing 4-1. Notice how the parameters in the script correspond to the fields in the WebLogic Administration Console. The script illustrates the values for `startcrm.sh` script parameters for running the samples.

Note: The port number is `7101`. You can change the port number; however, if the port number is changed make sure to change it in the corresponding field in the Gateway configuration CRM1 pane of the WebLogic Administration Console.

Figure 4-1 Fields for the CRM



Name:	CRM1
Status:	Up
Listen Address:	mainframe
Listen Port:	9002
Logical Unit:	BEAAPPL1
Stack Type:	vtm28

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.

Listing 4-1 Command to Run the CRM

```
$CRMDIR/bin/CRM //127.0.0.1:7101 CRM1 < /dev/null > std.out
2>std.err &
```

Starting the CRM on z/OS or OS/390 MVS

On z/OS or OS/390 MVS, start the CRM by submitting the `CRMSTART` JCL that is installed with the CRM. The `CRMSTART` JCL must be modified for your environment. For information about modifying the `CRMSTART` JCL, see the *BEA WebLogic Java Adapter for Mainframe Configuration and Administration Guide*.

As you modify the `CRMSTART` JCL, make sure that you note the following parameters in the value of the `STARTCMD` parameter in the JCL. These parameters correspond to fields in the WebLogic Administration Console. These values must be the same in the JCL and in the WebLogic Administration Console.

- The machine address where the CRM will run
- The port number on which the CRM will listen
- The name by which the Gateway will refer to the CRM

Compare Figure 4-1 with the JCL in Listing 4-2. Notice how the parameters in the JCL correspond to the fields in the WebLogic Administration Console. The JCL illustrates the values for `STARTCMD` parameters for running the samples.

- The machine where the CRM will run is `myhost` in this sample. You must replace `myhost` with the hostname or IP address of your mainframe to allow IP-based communication from the Windows or Unix machine where WebLogic Server is running. You may verify this parameter with the `ping` command on Windows or Unix.
- The port number is `7101`. You can change the port number; however, if you change the port number, make sure to change it in the corresponding field in the Gateway configuration CRM1 pane of the WebLogic Administration Console.

Note: BEA recommends that you do not change the CRM name from CRM1, because this name for the CRM is preconfigured for all of the samples.

Listing 4-2 The `STARTCMD` parameter in the `CRMSTART` JCL

```
//  SET STARTCMD='"//myhost:7101" CRM1'
```

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 *BEA WebLogic Java Adapter for Mainframe Configuration and Administration Guide*. In addition to defining an Logical Unit for the CRM to use, complete the following steps. The file `VTAMDEFINITION` in the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/explicitAPPC/mainframe/source` directory contains examples of the necessary VTAM definitions and configuration. See your VTAM specialist for assistance.

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 `APPCMxx 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 WebLogic JAM configuration. For this example, the name is `TOUPPER`.

Step 3: Update the WebLogic JAM Configuration File

On the machine where the Gateway is installed, update the WebLogic JAM configuration file from the command prompt by completing the following steps:

1. Locate the `jamconfig_BATCH.xml` file under the following directory:

   `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples`

2. Copy `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 WebLogic JAM Gateway

Most configuration tasks were preconfigured or were completed during the installation process by the installer program. For additional information about configuring WebLogic JAM, see the *BEA WebLogic 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:

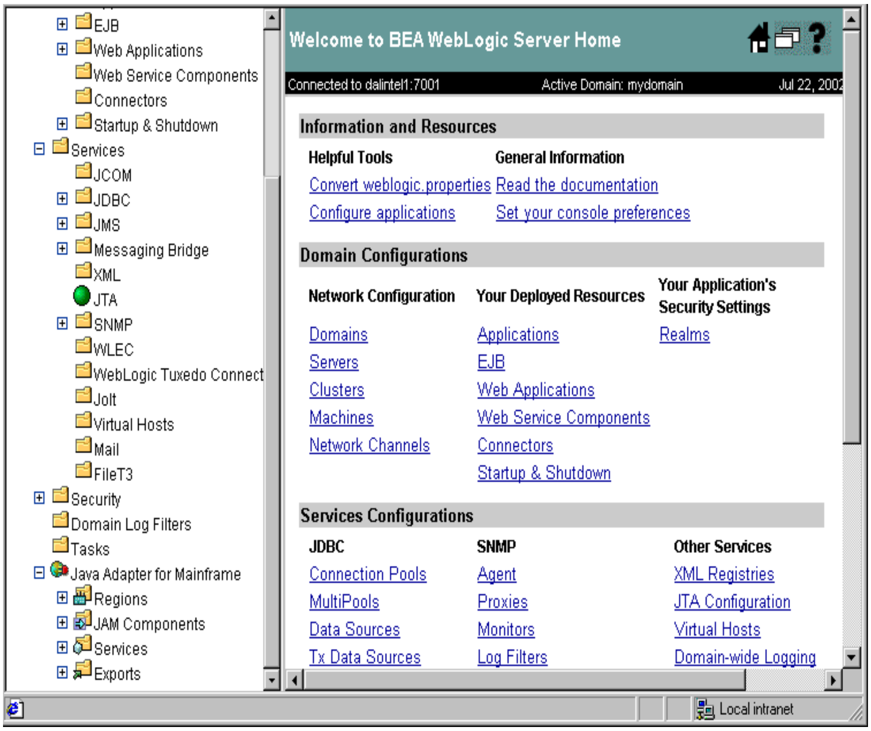
   - user: system

     This user name cannot be changed.

   - password: security

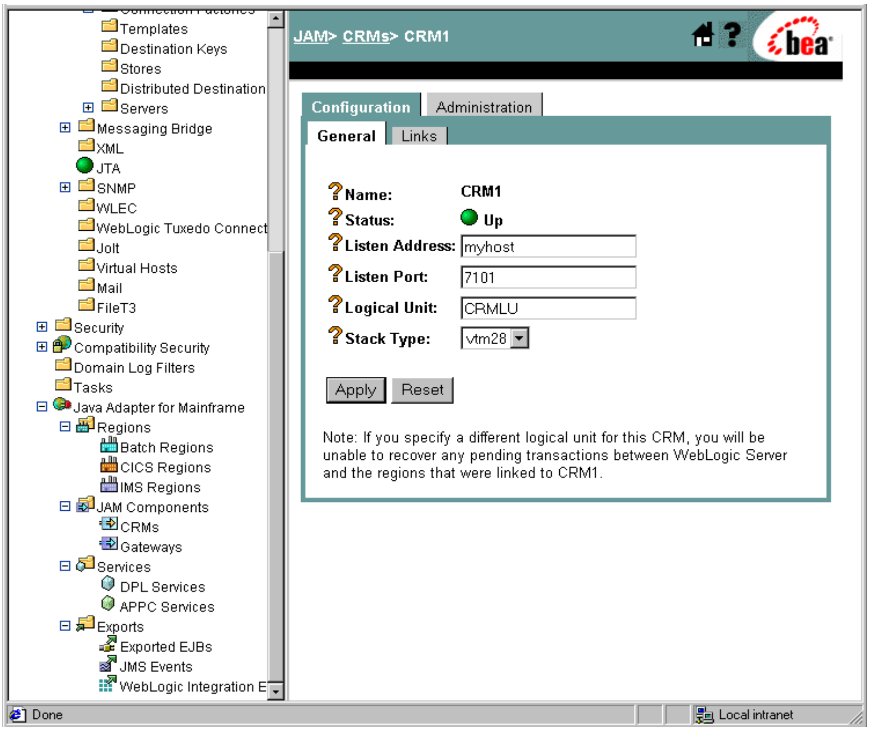
     To change the password, see the BEA WebLogic Server documentation.

The WebLogic Administration Console displays.



3. To configure the CRM to the WebLogic 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.



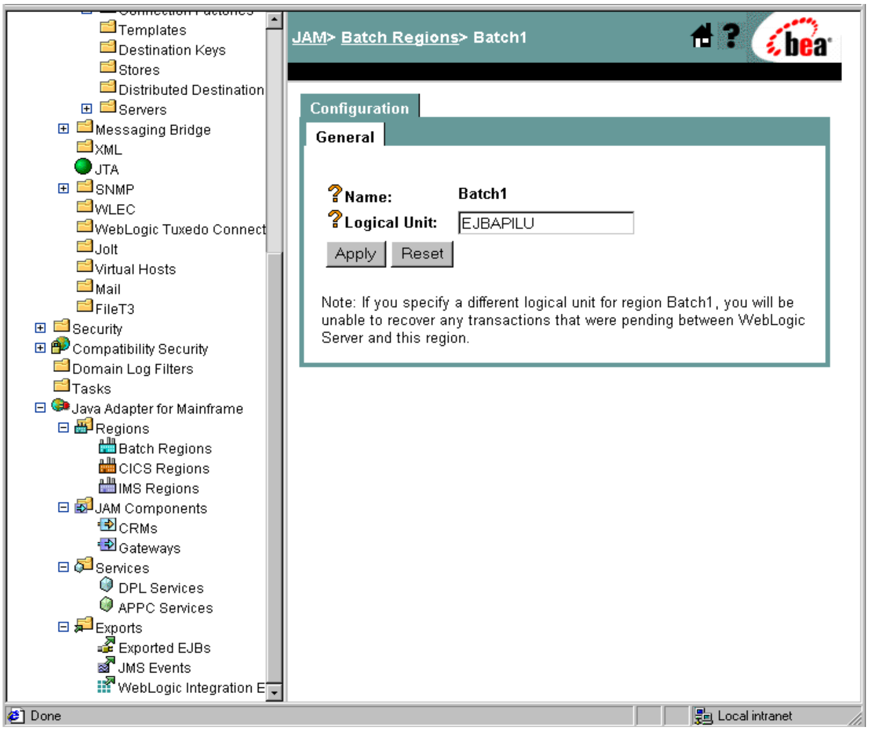
Name: CRM1
Status: Up
Listen Address: myhost
Listen Port: 7101
Logical Unit: CRMLU
Stack Type: vtm28

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).

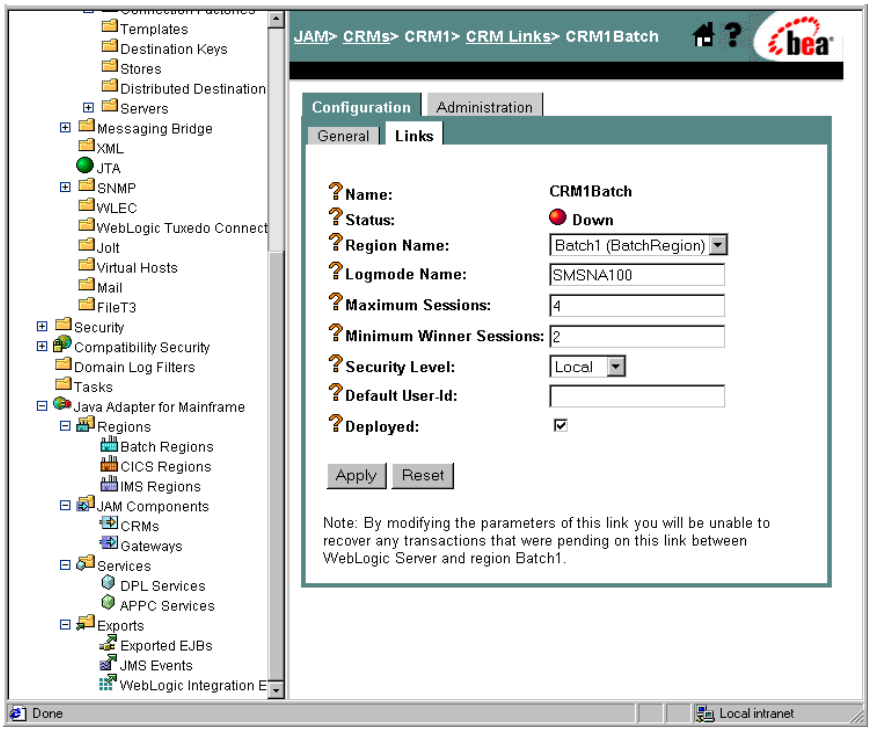


Name: Batch1

Logical Unit: EJBAPILU

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.

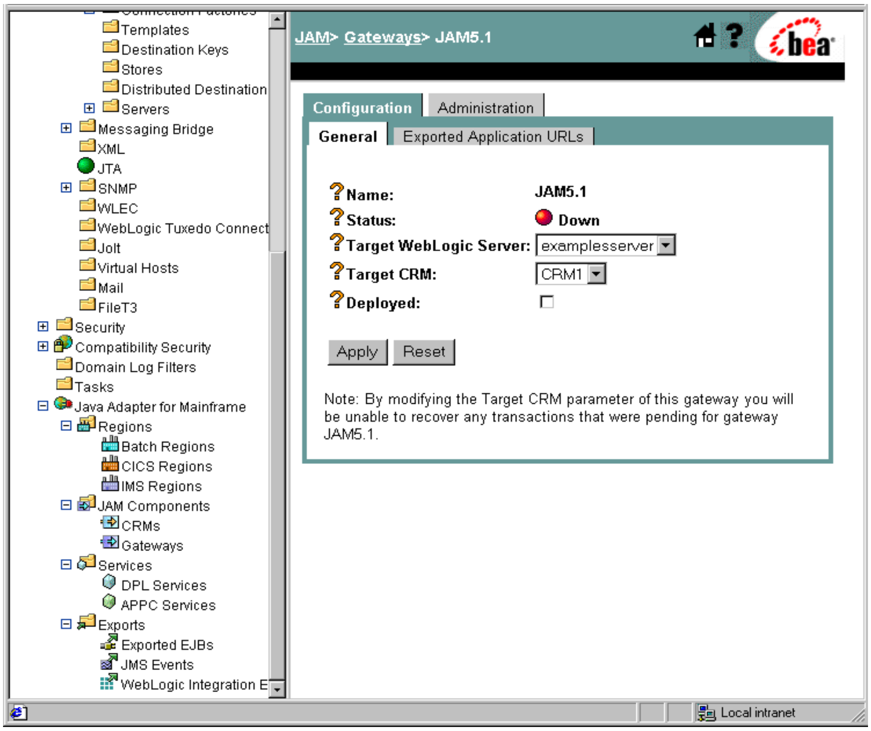
c. Click **Batch Regions** at the top of the pane. In the new window, click **CRM1Batch**. On the **Links** tab, check **Deployed** and click **Apply**.



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:	✓

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.

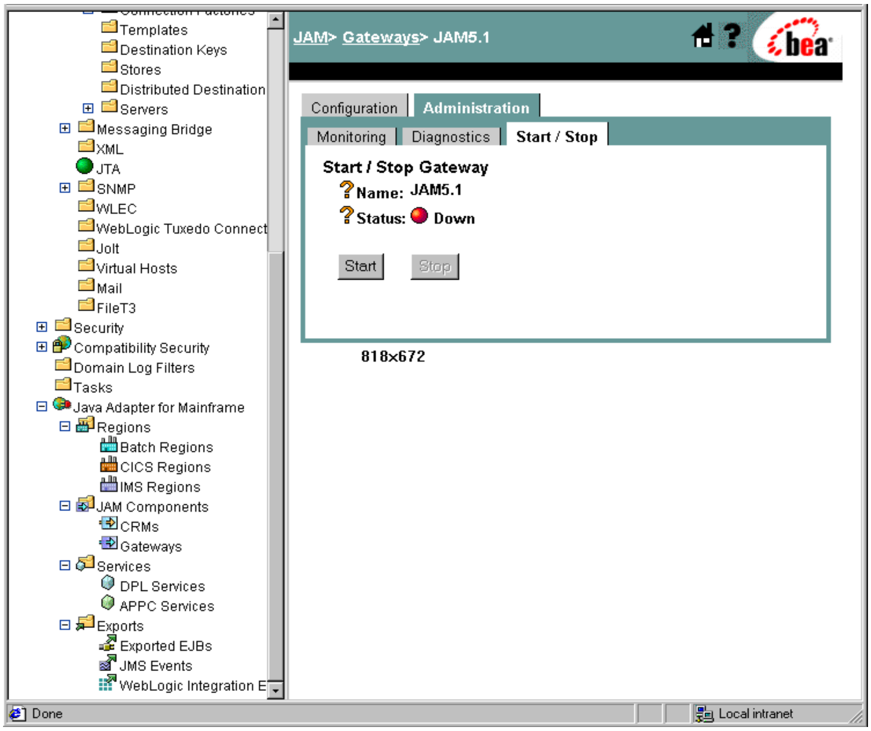
d. In the left pane, click **Gateways**. Click **JAM5.1** in the right pane. On the **General** tab, check **Deployed** and click **Apply**.



Name: JAM5.1
Status: Down
Target WebLogic Server: examplesserver
Target CRM: CRM1
Deployed: ☐

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



Start / Stop Gateway

Name: JAM5.1

Status: Down

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 the "Preparing to Use the Explicit APPC Sample" 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 WebLogic 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 WebLogic 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 WebLogic JAM configuration, in this case `TOUPPER`.

In the WebLogic 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 WebLogic 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 WebLogic JAM Gateway. The EJBAPI issues a receive to retrieve the response data.

Sample Files

The files for the WebLogic JAM side of the are installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/explicitAPPC/
gateway
```

The following table lists the samples files and their purpose:

Table 4-1 Files for the WebLogic JAM Application

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.
`toupperServer.egen`	eGen script that generates the `Chardata.java` DataView class, the `ToupperServer.java`, `ToupperServerBean.java`, `ToupperServerHome.java` bean classes, and the `ToupperServer-jar.xml` and `wl-ToupperServer-jar.xml` 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 `ToupperServerBean`. The business logic of servicing the requests from the mainframe is actually implemented in this class.
`build.cmd`	Script that builds the `Chardata`, `ToupperServer`, `ToupperServerBean`, `ToupperServerHome`, and `ExtToupperServerBean` classes. It assembles the classes along with the necessary deployment descriptors to a `.jar` file. The resulting `JAM_ToupperServer.jar` file is in the `<BEA_HOME>\<JAM_INSTALL_DIR>\config\examples\applications\examples\explicitAPPC\gateway` directory.
`build.sh`	Unix script that builds the `Chardata`, `ToupperServer`, `ToupperServerBean`, `ToupperServerHome`, and `ExtToupperServerBean` classes. It assembles the classes along with the necessary deployment descriptors to a `.jar` file. The resulting `JAM_ToupperServer.jar` file is in the `<BEA_HOME>/<JAM_INSTALL_DIR>/config/examples/applications/examples/explicitAPPC/gateway` directory.
`ToupperServer-jar.xml`	Deployment descriptor generated by `toupperServer.egen`.

File Name	File Purpose
`wl-ToupperServer-jar.xml`	WebLogic deployment descriptor generated by `toupperServer.egen`.
`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 installed in the following directory:

```
<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/explicitAPPC/
mainframe/source
```

The following table lists the samples files and their purpose:

Table 4-2 Files for Mainframe Application

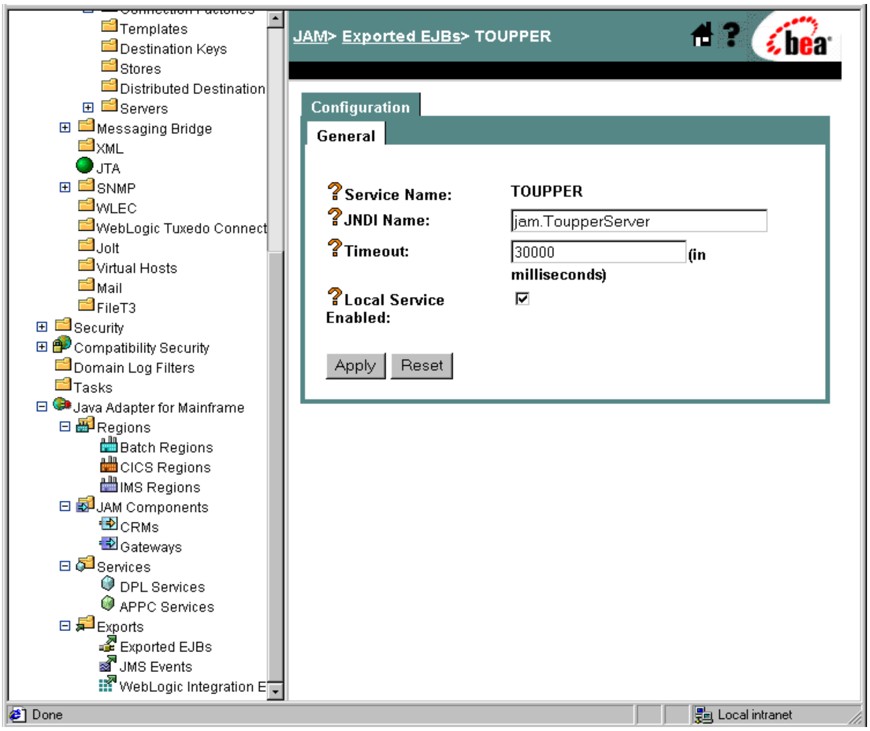
File Name	File Purpose
`ASEJBAPI`	JCL that assembles the WebLogic EJB API interface module `EJBAPI`.
`CLCLIENT`	JCL to compile and link the batch client `WLCLIENT`.
`EXCLIENT`	JCL to execute the batch client `WLCLIENT`.
`EJBAPI`	The WebLogic EJB API interface module.
`VTAMDEFINITION`	Contains sample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

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

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.



Service Name: TOUPPER
JNDI Name: jam.ToupperServer
Timeout: 30000 (in milliseconds)
Local Service Enabled: ☑

Step 2: Set the Environment

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

- 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: Generate and Build Source (Optional)

The WebLogic JAM samples provide generated source. The samples also provide classes to run the samples. If you want to see how the source is generated and the classes are built, change to the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/explicitAPPC/gateway` directory and complete the following steps.

Warning: Using the following options will overwrite files that are installed with the WebLogic JAM samples.

- Run the `build.cmd (.sh)` script to build the bean classes and put them in the `JAM_ToupperServer.jar`.
- Run `egencobol` to use the eGen Application Code Generator on `toupperServer.egen` to generate the code for the `ToupperServer` bean and its deployment descriptors.

  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 *BEA WebLogic Java Adapter for Mainframe Programming Guide*.

Step 4: 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 from the `<BEA_HOME>/<JAM_INSTALL_DIR>/samples/examples/explicitAPPC/mainframe/source` directory in to the source PDS that you created:
   - `ASEJBAPI`
   - `CLCLIENT`
   - `EJBAPI`
   - `EXCLIENT`
   - `WLCLIENT`
4. In the `ASEJBAPI` JCL, make the following changes:
   - Change the `JOB` statement.
   - Change the `YOURHLQ.OBJECT` to the PDS that you allocated for objects.
   - 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:
   - Change the `JOB` statement.
   - Change the `YOURHLQ.OBJECT` to the PDS that you allocated for objects.
   - Change the `YOURHLQ.SOURCE` to the PDS that you allocated for source.
   - Change the `YOURHLQ.LOAD` to the PDS that you allocated for executables.
   - 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:
   - Change the `JOB` statement.
   - Change the `YOURHLQ.LOAD` to the PDS that you allocated for executables.
   - 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`.

Index