Developing and Deploying JMS Enabled CICS Applications

Version V1.0

August 2015

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

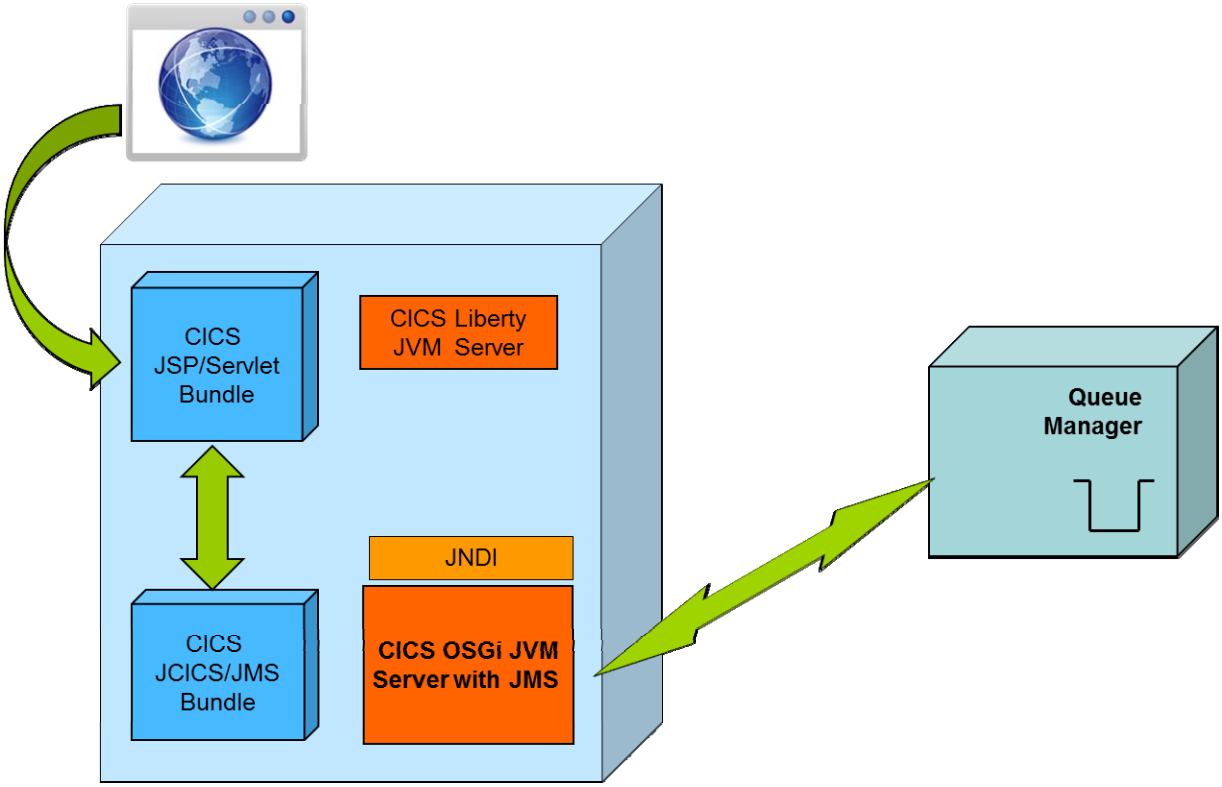
Exercise Overview

The goal of this exercise is to provide an opportunity to gain knowledge and experience in developing and deploying CICS Java Messaging Service (JMS) enabled applications. Even though there is little actual coding involved, completing this exercise will provide an overview and understanding of the basic coding requirements for Java JMS in CICS. Completing this exercise will also provide experience with using the tools to deploy and test JMS application in CICS.

- The exercise starts by explaining JMS terms and some of the basic Java programming concepts used in developing JMS applications in general.
- MQ Explorer will be used to create the JNDI binding file used by CICS JMS support to associate application names for queue manager and queues with the actual names and connection information for the queue manager and queues.
- CICS Explorer will be used to review a simple application that accesses both CICS and JMS resources. This application consists of a JSP/Servlet component (subsequently referred to as the JMS web application and a JMS enabled CICS program (subsequently referred to as the JMS CICS application). A few minor changes are required to make the JMS web application portion of the application in order to make its execution specific to your assigned team. Once these changes are made, the JMS web application will be deployed to CICS and installed for execution.
- Various test scenarios will be performed including scenarios that induce errors. CICS's Execution Diagnostic Facility transaction CEDX will be used to review the flow of the CICS application as it executes.

The diagram below illustrates the flow and application artifacts involved in this exercise.

- A user at a web browser will enter an URL which will invoke a JMS web application that is running in a CICS Liberty JVM server.
- The JMS web application will collect information from the end-user and uses it to populate a CICS container.
- This container is passed via a CICS Link to the CICS JMS application which is running is a CICS OSGi JVM server which has IBM MQ JMS support installed.
- The JMS CICS application will perform the requested function (PUT or GET) and return the results in another container back to the JMS web application
- The JMS web application will display the results in the browser.



General Lab Information and Guidelines

- Information required to complete this exercise will be provided on a 'worksheet' prior to the start of this exercise. Refer to this worksheet for which user identity and password are to be used and for other values, for example:

  - ✓ Any time a reference is made to TEAMXX, teamxx, XX, xx, or T0XX appears in the instructions; please replace the XX or xx characters with your assigned team identifier number (01 – 20). Some of these occurrences of these strings with X's are case sensitive so be sure to not change the case of other characters.

  - ✓ Screen shot may contain references to QML1, MPX1 or 1417. You should use your assigned queue manager name, queue manager port or LPAR name from the worksheet.

  - ✓ As a reminder, when a value from your worksheet should be used, the values in the instructions will be in red rather than black.

  - ✓ ***Bold italicized*** text indicates values that need to be entered on a screen.

  - ✓ *Italicized* text indicates values that are constants or names that appear on a screen.

  - ✓ **Bold** text indicates the name of buttons or keyboard keys that need to be pressed.

This exercise requires a workstation with IBM CICS Explorer installed along with the CICS Explorer SDK. IBM MQ V8 at fix pack level 2 is also required on the workstation in order to have access to the JMS OSGi bundles provided in fix pack 2. This environment is already provided for you.

The z/OS LPAR has CICS TS 5.2 with CICS APAR PI32151 and IBM MQ V8 with IBM MQ APAR PI32151 installed. These APARS added support for JMS applications in CICS as wells as Open Services Gateway Initiative (OSGi, see URL http://www.osgi.org/About/HomePage) support for JMS in IBM MQ V8.

The z/OS LPAR have a CICS TS 5.2 region with two CICS JVM Servers. One JVM server is named CSCWLP which has been configured as a Liberty JVM server (see the Appendix) and the other is CSCJVM which has been configured as an OSGi JVM server with JMS support enabled. The CICS regions use a MQCONN resource to connect to a local queue manager.

Part 1 - Introduction to Java Message Service

This section simply provides an overview of Java Message Service (JMS) and JMS related terms as well as a simple introduction or explanation of JMS coding practices.

Java Message Service

First, JMS is an application programming interface (API) that describes a standard programming interface for sending and receiving messages between applications. The Java classes and methods described by the JMS specification when used by a JMS applications are independent of the underlying messaging services provider (e.g. IBM MQ, etc.) and therefore any application that uses JMS should be able to run with any messaging provider product that adheres to the JMS specification.

Java Naming and Directory Interface (JNDI)

JMS applications require access to a Java Naming and Directory Interface (JNDI) service. A JNDI service provides common naming and directory services to Java clients so they can look up or obtain information simply by specifying a name (a JNDI name) of a resource. So rather than statically coding properties of a resource, Java clients can use a JNDI service to look up and obtain information about resources while executing (i.e. runtime or dynamic binding). JNDI defined resources can range from data bases, CICS regions, IMS regions, MQ queue managers, enterprise java beans and so on. A Java client uses a name (JNDI name) that has been previously defined in the JNDI name space to locate a resource and gain access to its properties. These properties are then used by the container in which the Java client is executing to access the resources on behalf of the Java client.

JMS JNDI Usage

From a JMS perspective, the resources defined to JNDI are known as ‘factories’ or as ‘destinations’. A JMS *connection factory* defines the connection properties (e.g. server or client bindings, queue manager, name, host, port, etc.) that are required for connecting to a specific queue manager. A JMS *destination* associates the name of a specific MQ queue or topic with a JNDI name. Also available but seldom used since the introduction of unified/domain independent connection factory in JMS 1.1 are *queue connection factories* and *topic connection factories.* These factories provided specialized queue-related or topic-related properties along with connection properties.

Use of JNDI lookups of JMS resources means that JMS applications can be written without providing specific names for a queue manager and/or queues and without specifying specify explicit connection properties. In other words, a JMS application can use the same internal names (JNDI names) for queue managers, queues and topics during developing, test and production therefore allowing an administrator to associate these internal names with the actual queue manager connection details, MQ queue or alias names, etc. required at runtime.

Using a non-CICS Liberty configuration file (server.xml) as an example (see below), there is one JMS connection factory for a local queue manager (*LMQM*) and 4 JMS connection factories for client connection queue managers. One JMS queue is defined. A JMS application can do a look up of a JNDI name *jms/QML1* and the connection details for this queue manager are now available. The same application can do a lookup of JNDI name *jms/Q1* and now the target queue has been identified. The application has no prior knowledge of this information and no modifications are required.

```
*server.xml

<jmsConnectionFactory jndiName="jms/LMQM">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" queueManager="LMQM" transportType="BINDINGS"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML1">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx1" port="1417" queueManager="QML1"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML2">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx2" port="1418" queueManager="QML2"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML3">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx1" port="1419" queueManager="QML3"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QML4">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx2" port="1420" queueManager="QML4"/>
</jmsConnectionFactory>

<jmsQueue id="Q1" jndiName="jms/Q1">
    <properties.wmqJms baseQueueName="SYSTEM.DEFAULT.LOCAL.QUEUE"/>
</jmsQueue>

Design  Source
```

A typical JMS application would start by establishing addressability to the JNDI namespace. Addressability to the JNDI service is done by obtaining an initial context object from the JNDI service.

```
//Create the JNDI initial context environment
Hashtable<String, String> environment = new Hashtable<>();
environment.put(Context.PROVIDER_URL, "file:///u/" + commarea.getUserid().trim() + "/jndi/");
environment.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.fscontext.RefFSContextFactory");


// Instantiate the initial context
Context context = new InitialContext(environment);
```

Once the initial context to the name space has been obtained, this context can be used to obtain a *connection factory* for use in establishing the connection to the queue manager as well as a reference to the *destination* or queue.

```
// Lookup the connection factory using the value of variable "queueManager"  (Qmgr in COMMAREA)
// This name must be defined in the .bindings file in the directory specified by PROVIDER_URL
ConnectionFactory cf = (ConnectionFactory) context.lookup(queueManager);

 // Lookup the destination (queue) using value of variable "queueName" (QName in COMMAREA)
 // This name must be defined in the .bindings file in the directory specified by PROVIDER_URL
 destination = (Destination) context.lookup(queueName);
```

Once the connection factory and any destinations are obtained, they can be used by the Java JMS code to connect to the queue manager, establish a session and do PUTs and GETs. The other JMS application interfaces will be explored later in this document.

Part 2 - JMS Applications and the MQ Explorer

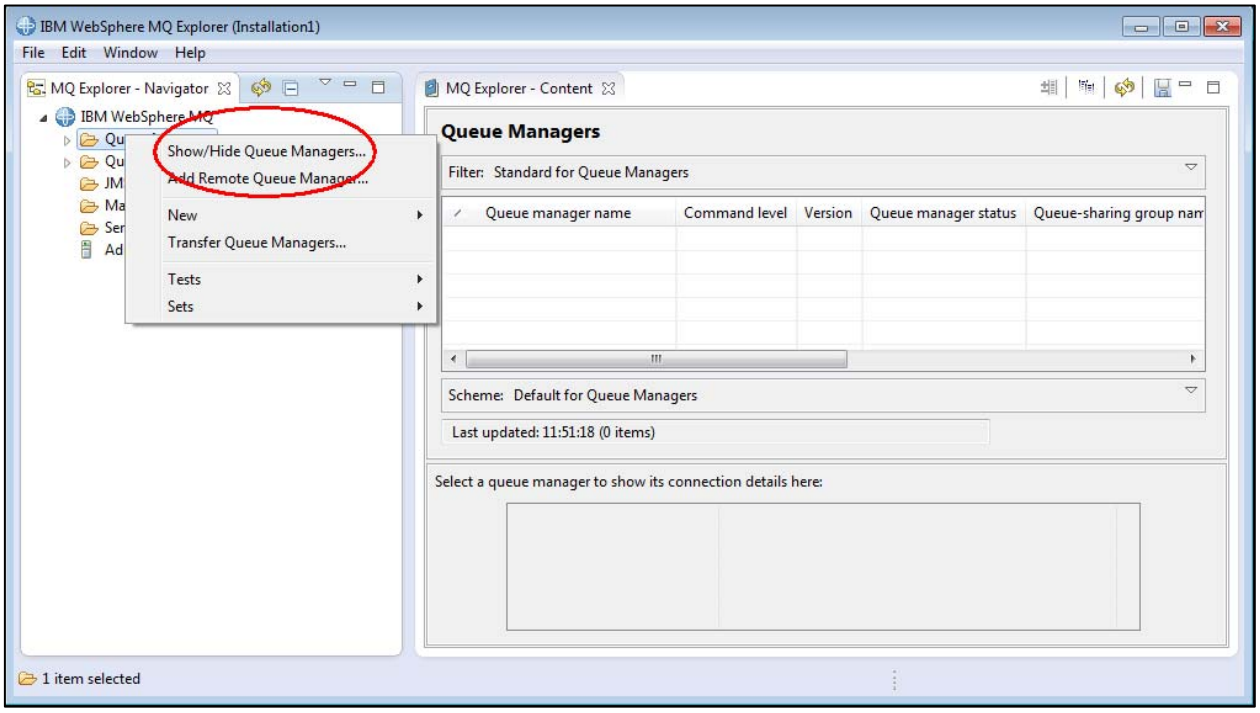
As described earlier, Java JMS applications perform an indirect lookup to a JNDI service provider using a JNDI name in order to locate queue manager and queue information. Since CICS only supports a file based JNDI service provider, a file must be created and made available to CICS JMS support by the application at execution time. This file has to be in a directory identified by the application and the file name must be ***.bindings***. A command line interface (JMSAdmin) is available for creating this bindings file but this exercise will use the MQ Explorer to create a bindings file for subsequent moving to an OMVS directory on z/OS.

This part of the exercise begins by using the MQ Explorer to create the ***.bindings*** file. The first step is to configure a connection to the queue manager using its name, host and port information provided on the worksheet.

____1. Double click the WebSphere MQ explorer icon on the desktop to start MQ Explorer.

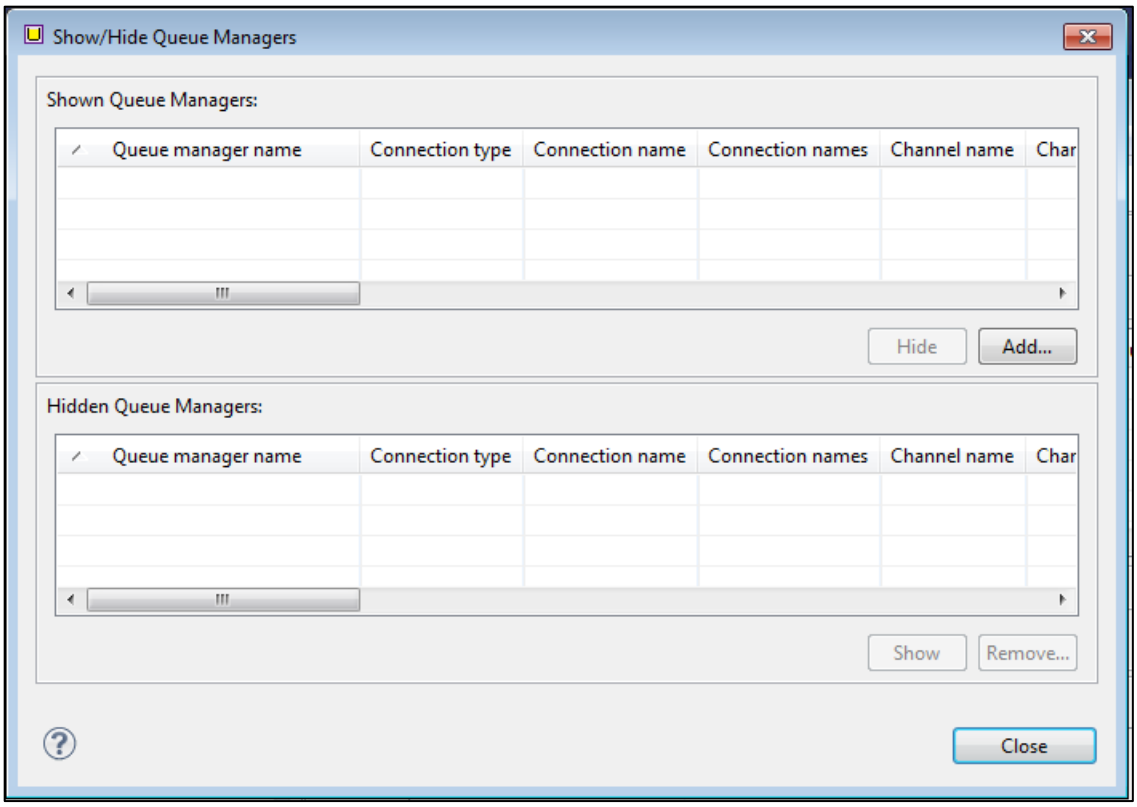
It may take a while for the MQ Explorer to fully initialize, so please be patient

____2. Right click on *Queue Managers* and select *Show/Hide Queue Managers*.

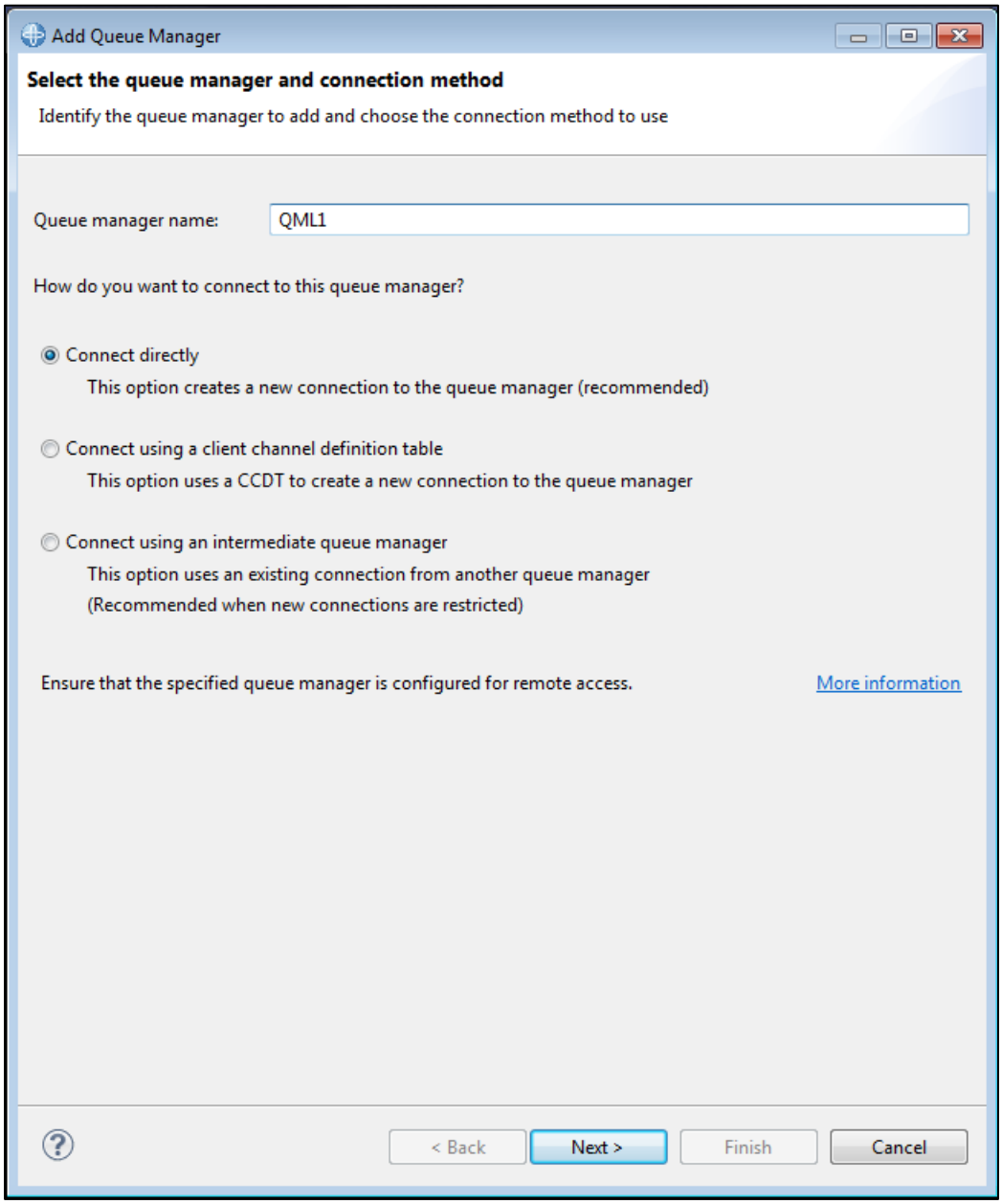


Tech-Tip: Eclipse based development tools like MQ Explorer provide a graphical interface consisting of multiple views within a window. A view is an area in the window dedicated to providing a specific tool or function. For example in the window above, *MQ Explorer - Navigator* and *MQ Explorer - Content Repositories* are views that use different areas of the window for displaying information. At any time a specific view can be enlarged to fill the entire window by double clicking in the view's title bar. Double clicking in the view's title bar will restore the original arrangement. If a view is closed or otherwise disappears, the original arrangement can be restored by selecting Windows → Reset Perspective in the window's tool bar.

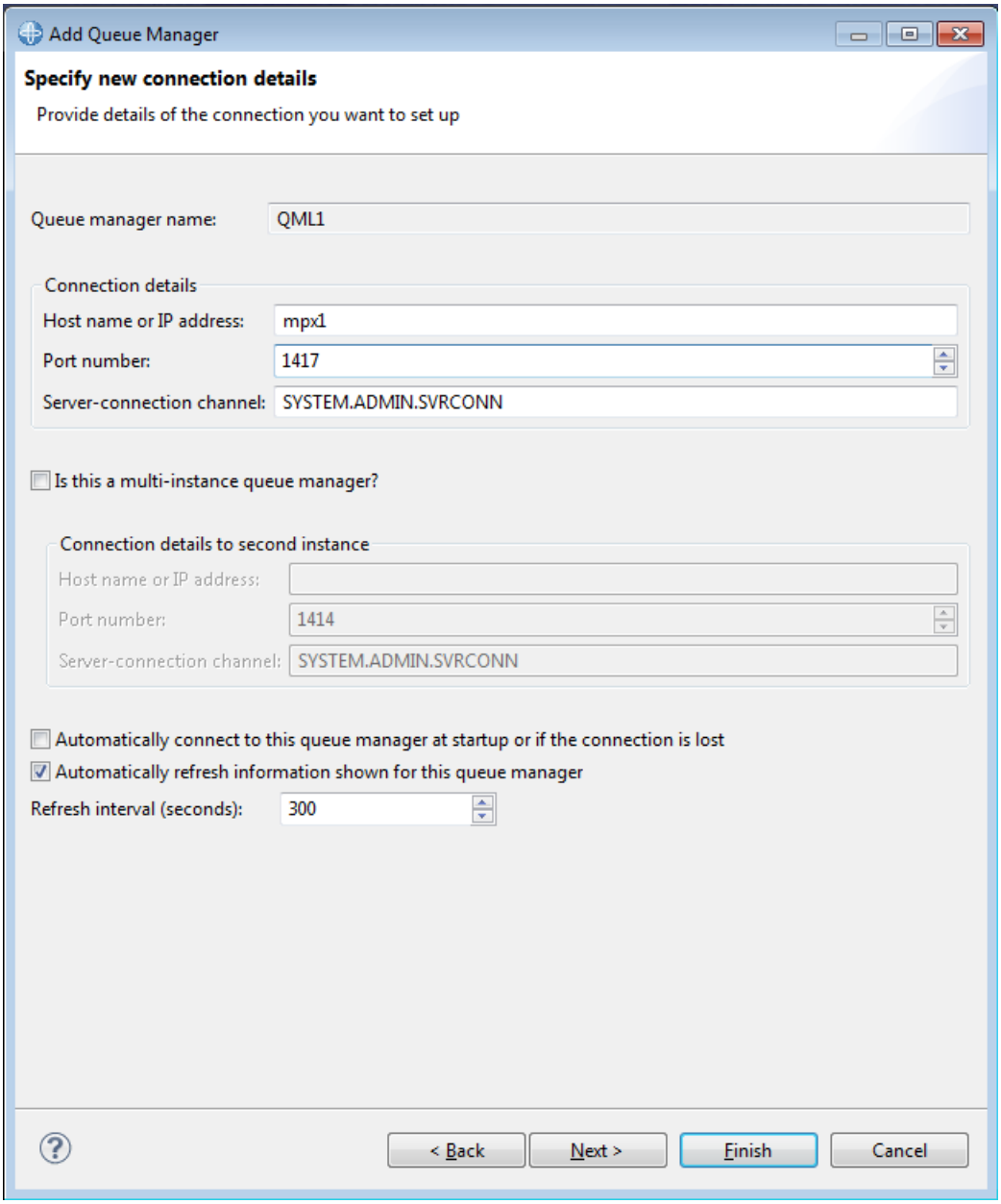
____3. The *Show/Hide Queue Manager* widow will now be displayed. Click on the **Add** button to continue.



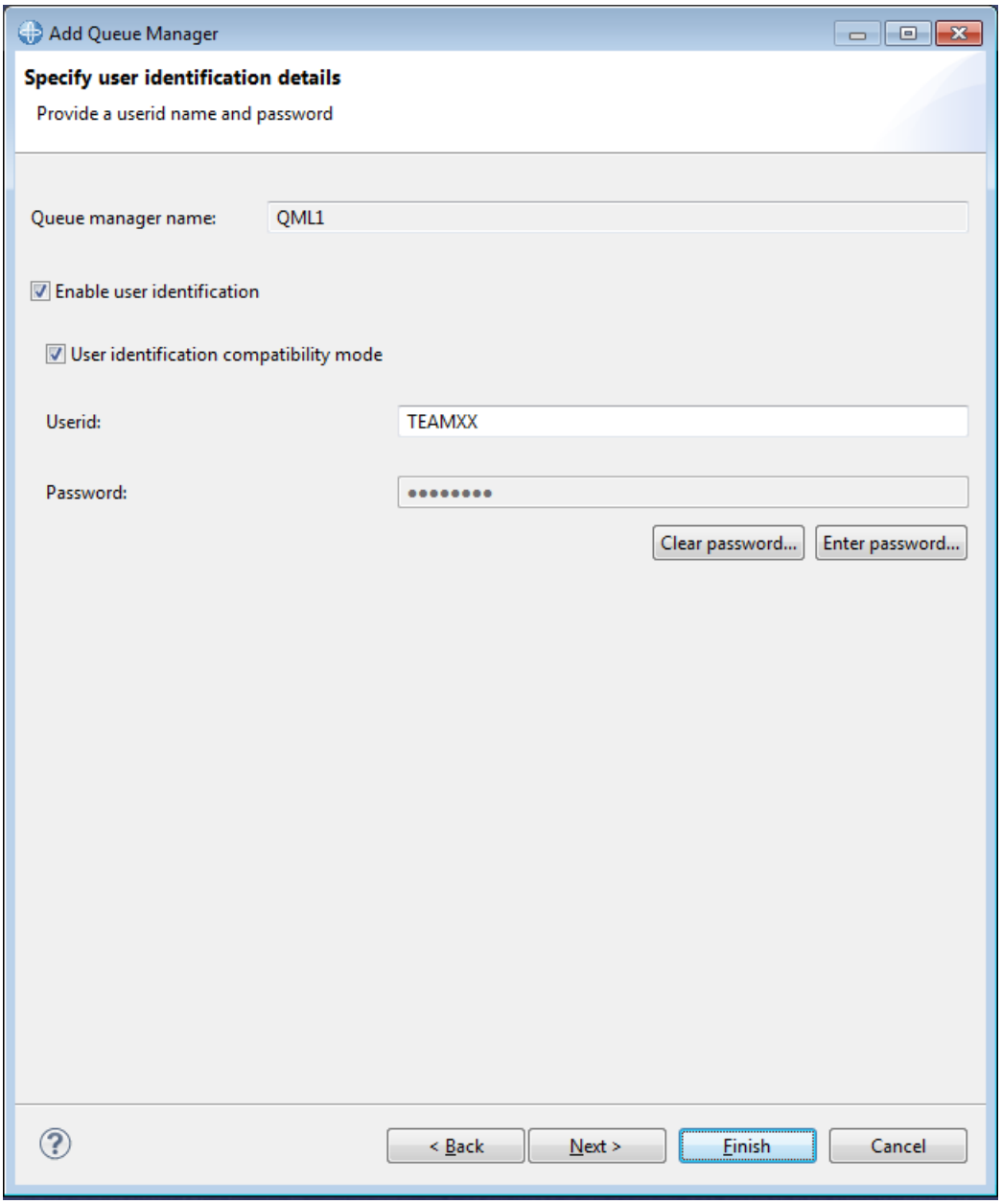
____4. Enter your assigned queue name (e.g. ***QML1)*** as the *Queue manager name* on the *Add Queue Manager – Select the queue manager and connection method* window and click **Next** to continue.



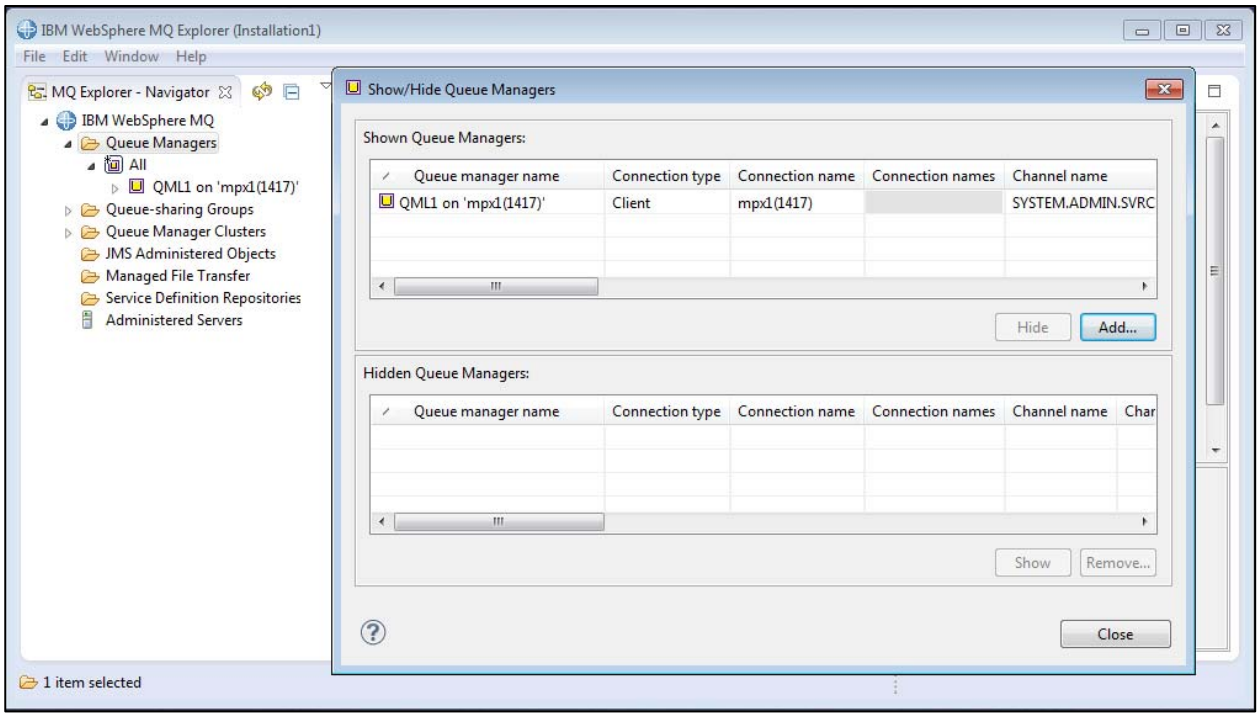
____5. Enter the host name of your assigned queue manager (e.g. ***mpx1)*** as the *Host name or IP address* and the host's IP address (e.g. ***1417)*** as the *Port number* on the *Add Queue Manager – Specify new connection details* window. Click the **Next** button twice to continue.



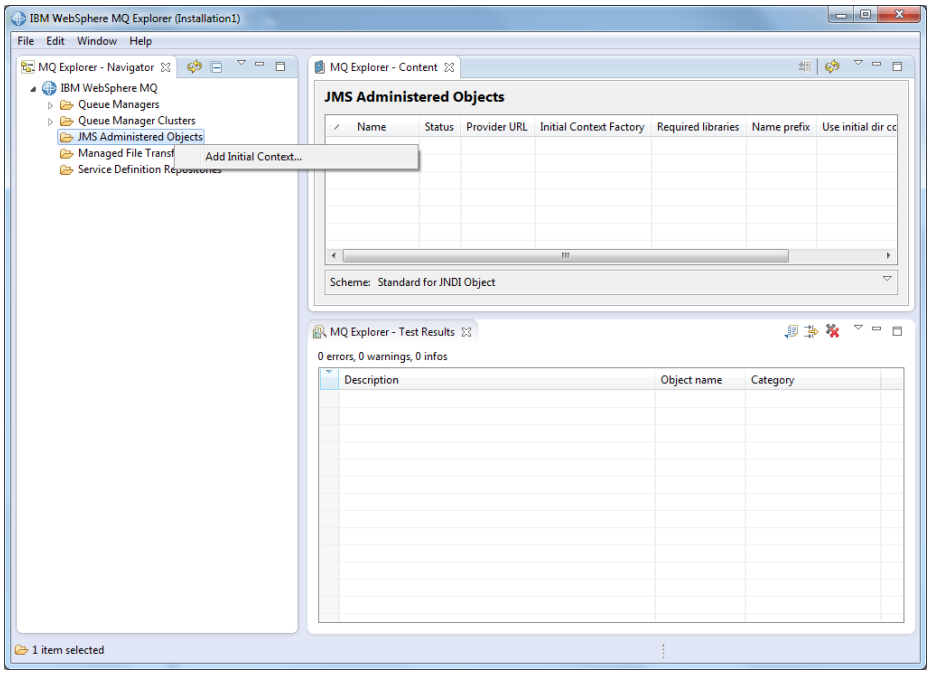
____6. The *Add Queue Manager – Specify security identification details* window should now be displayed. Check to box beside *Enable user identification* and enter in upper case your assigned user identifier in the *Userid* field and click the **Enter password** button. Enter the password in upper case and click the **OK** button. Click the **Finish** button to continue.



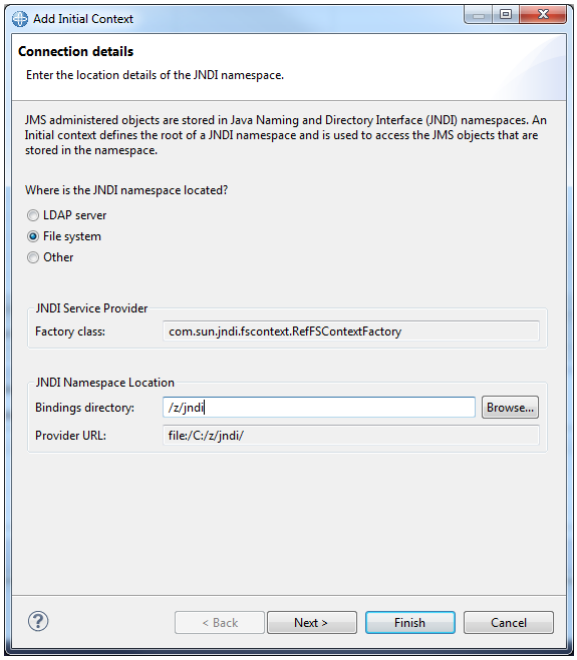
____7. The assigned queue manager (e.g. QML1) should appear in the *Queue Managers* list as shown below. Click the **Close** button.



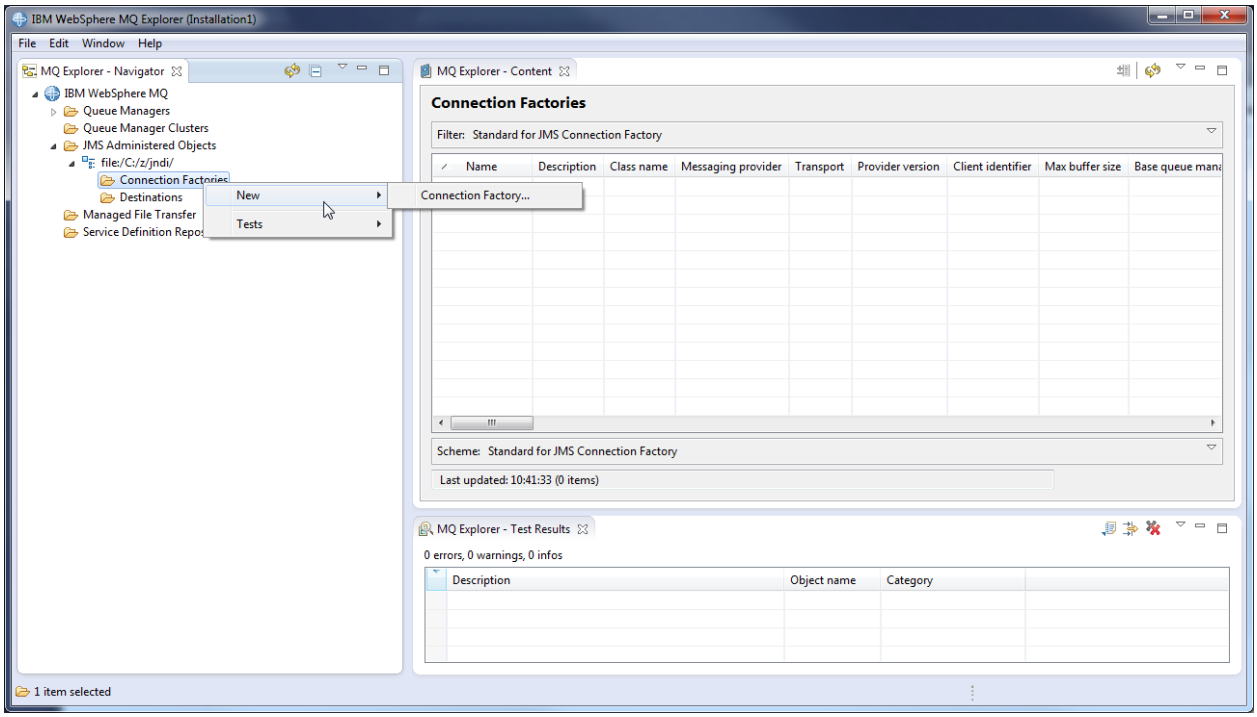
____8. Select *JMS Adminstered Objects* and right mouse button click (see below). Select *Add Initial Context* to continue.



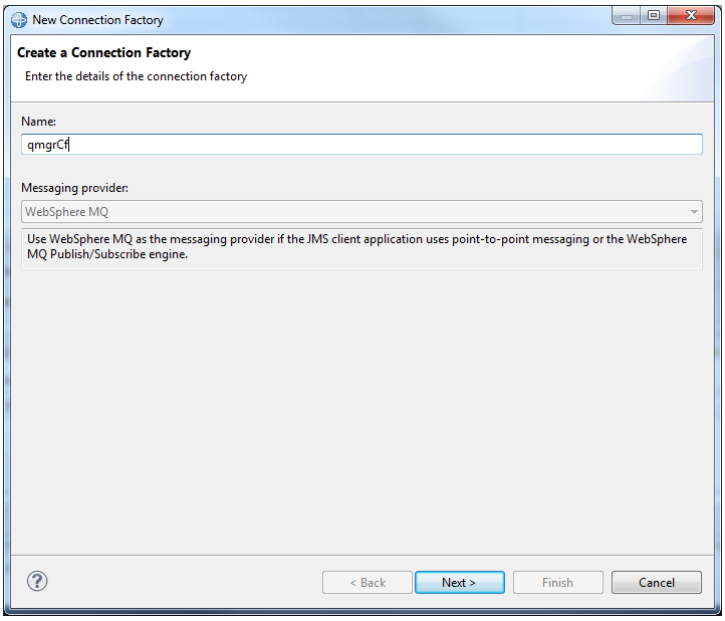
____9. On the *Add Initial details – Connection* details screen select the radio button beside *File System* and in the area beside *Binding directory* enter the name of the directory where the informaiton should be stored, *e.g. /z/jndi*. Click **Finish** to continue.



____10. Next expand the initial context folder just created in MQ Explorer and select *Connection Factories*. Right mouse button click and select *New –> Connection Factory* (see below).



____11. On the *New Connection Factory – Create a Connection Factory* screen enter ***qmgrCf*** as the Name of this connection factory. Click **Next** to continue. Note that the names of connection factories and destinations are case sensitive



Note, the application only supports a connection factory name of *qmgrCf* in a drop down list.

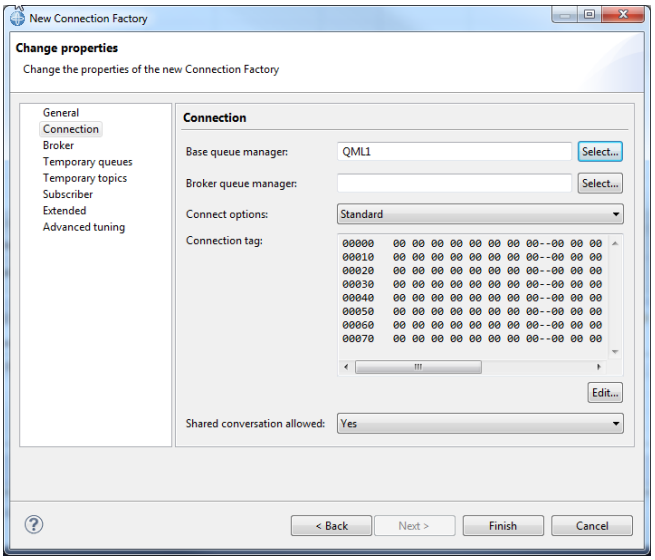
> Tech-Tip: The name of the connection factory will be used by a JMS application to in the lookup of the connection factory at runtime.
>
> ```
> ConnectionFactory connfactory = (ConnectionFactory)ctx.lookup("qmgrCf");
> ```

____12. Click **Next** on the *New Connection Factory – Create a Connection Factory* screen to take the default of a general *Connection Factory*.

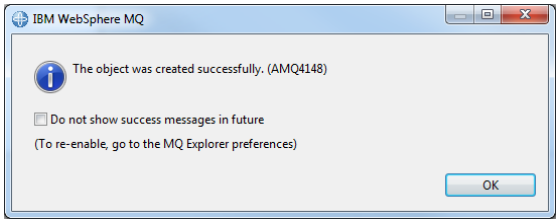
____13. Click **Next** on the *New Connection Factory – Create a Connection Factory* screen to take the default of *Bindings* for the *Transport*. (A transport type of *MQ Client* is not support in CICS).

____14. Click **Next** on the next *New Connection Factory – Create a Connection Factory* screen to continue.

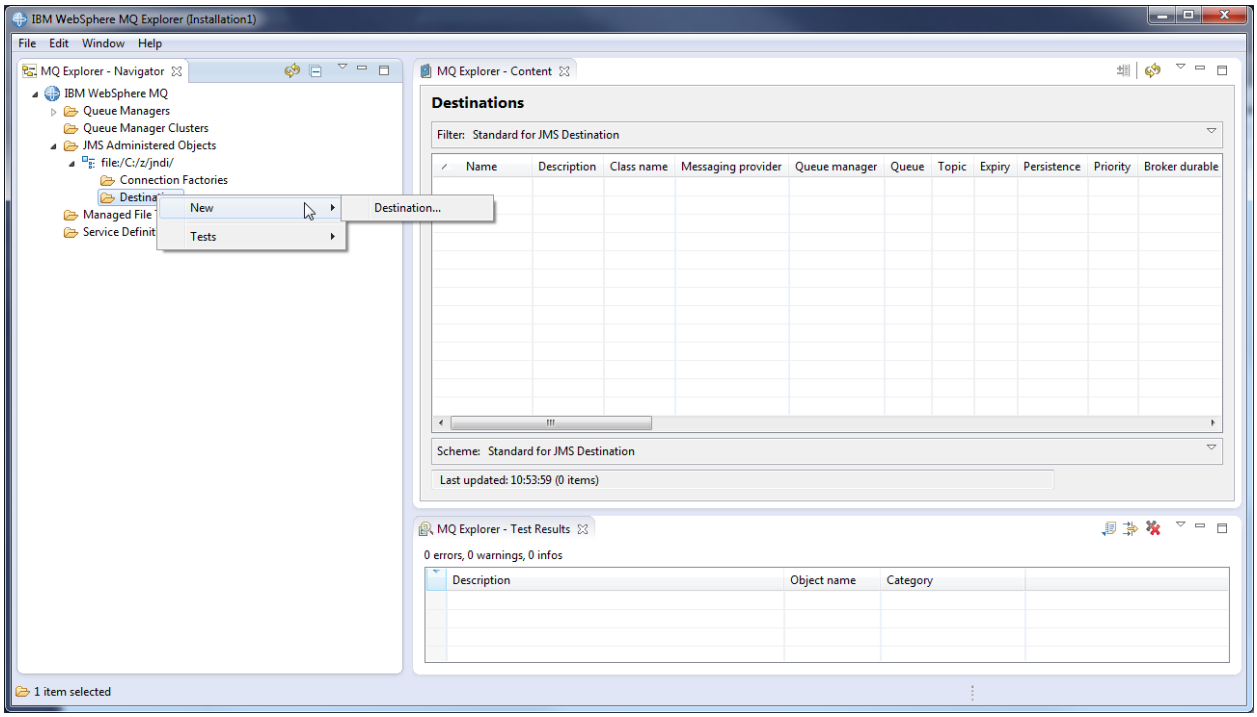
____15. On the *New Connection Factory – Change properties* screen select the *Connection* tab and use the **Select** button beside *Base Queue manger* to select the queue manager name (see below). Click **Finish** to continue.



Please note that the subsequent steps assume that the confirmation popup window (see below) has been disabled.



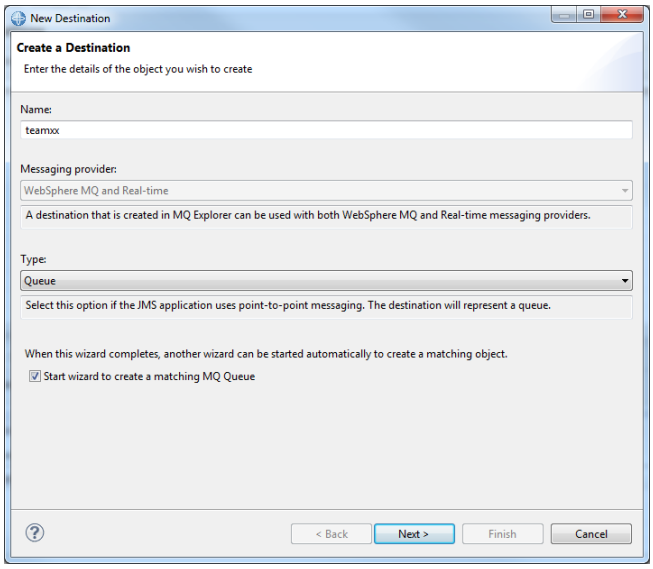
____16. Back on the main screen select *Destinations* and right mouse button click and select *New -> Destination* to continue (see below).



Tech-Tip: The information provided in the .bindings file is used by the CICS application to know what queue is actually to be used when the JMS application does a lookup for queues "teamxx" or "default".

```
Destination destination = (Destination)ctx.lookup("teamxx");
```

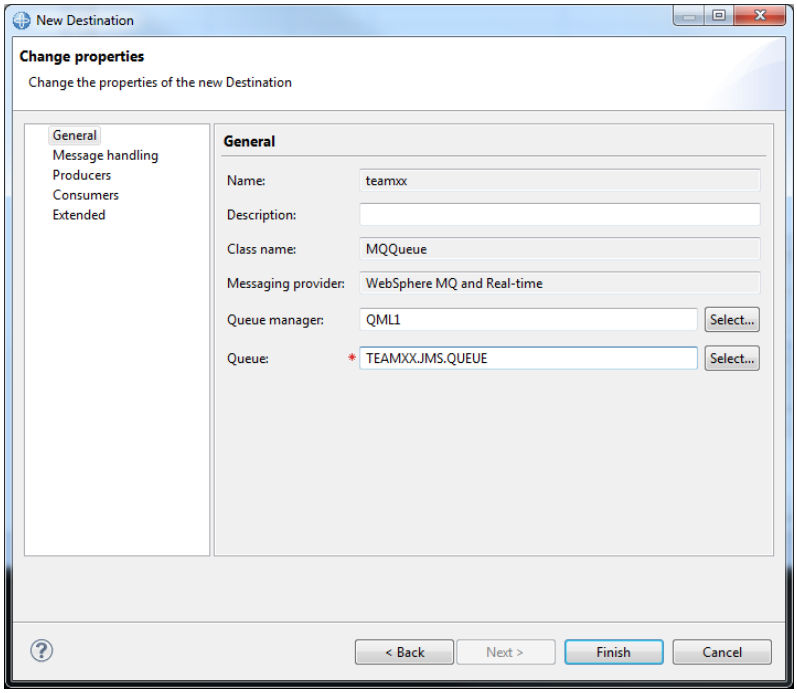
____17. On the *New Destination – Create a Destination* screen enter ***teamxx*** as the JNDI name for the new desination. Check the box beside *Start wizard to create a matching MQ Queue* and click **Next** to continue.



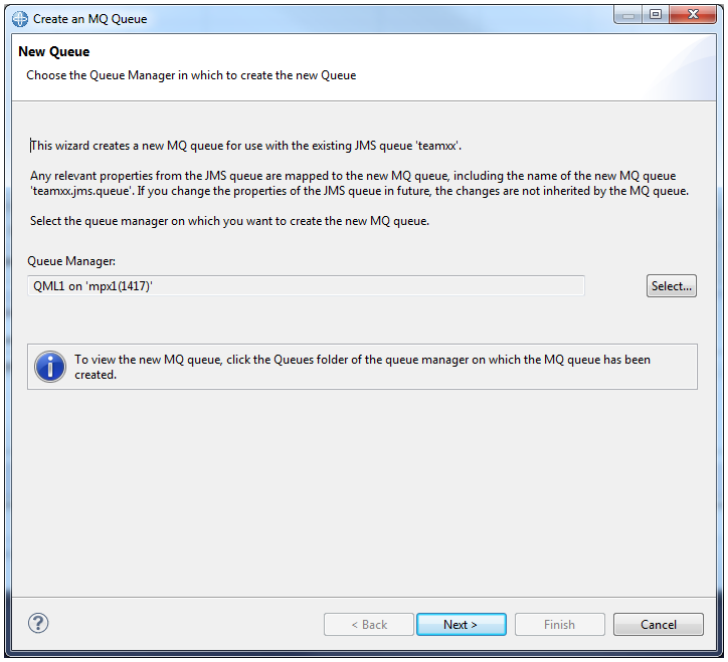
Note, the JSP only provides the values team01 through team20, teamxx and default in a drop down list.

____18. Click **Next** on the next *New Destination – Create a Destination* screen to continue.

____19. On the *New Destination – Change properties* screen use the **Select** buttons to specify the queue manager and and enter the name of queue, e.g. ***TEAMXX.JMS.QUEUE,*** which is to be associated with this JMS destination. Click **Finish** to continue.

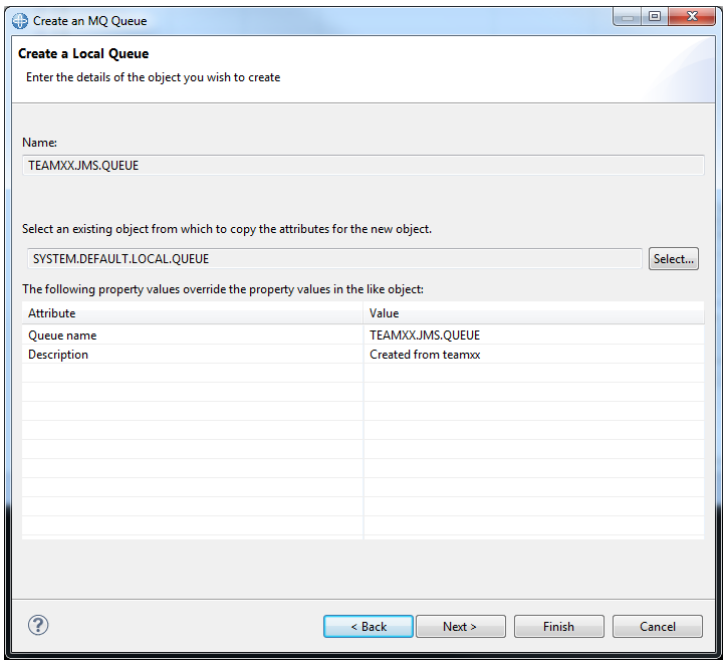


____20. Since the box to start the create a queue wizard was checked back in Step 17, the next screen displayed should be the *Create an MQ Queue – New Queue* screen, see below. On this screen click **Next** to continue.

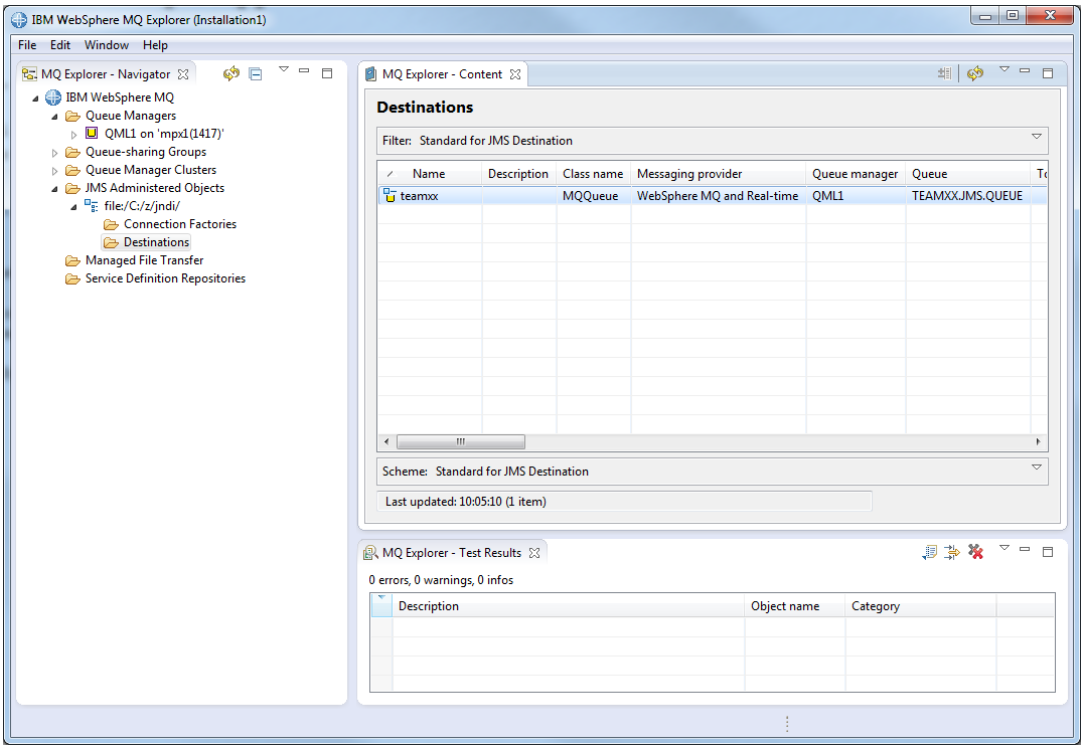


____21. Click **Next** on the next screen to select a type of local queue.

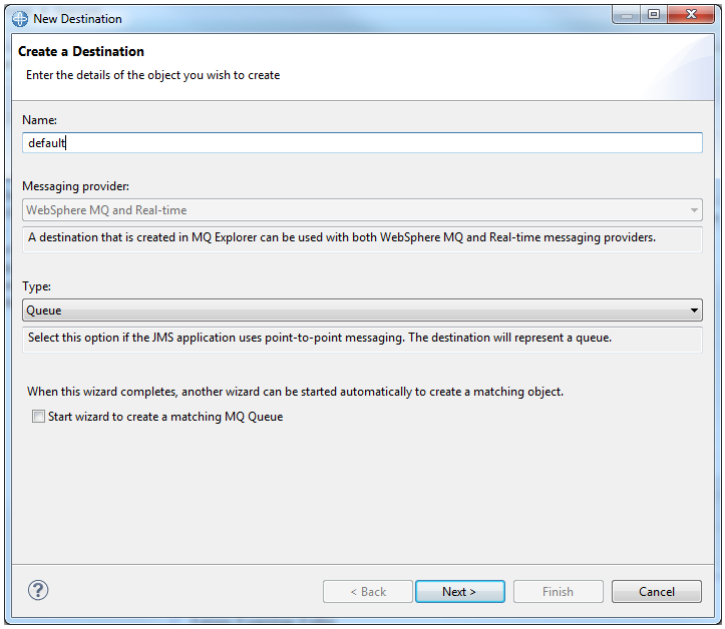
____22. Take all of the default values for queue properties by clicking on the **Finish** button on the *Create an MQ Queue – Create a Local Queue* screen.



____23. This action will redisplay the *Destination* pane with the new destination listed (see below). Another destination needs to be added so repeat Step 16 to restart the process.

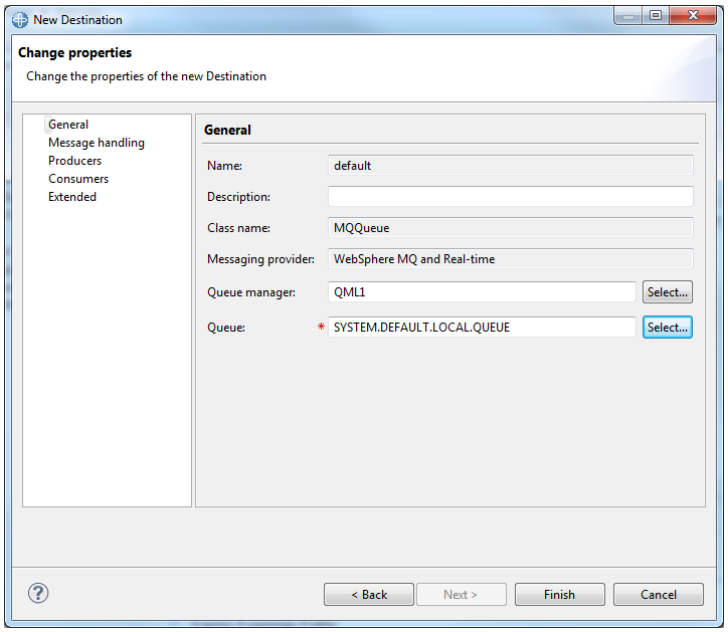


____24. On the *New Destination – Create a Destination* screen enter ***default*** as the JNDI name for the new desination. But do not check the box beside *Start wizard to create a matching MQ Queue.* Click **Next** to continue.



____25. Click **Next** on the next *New Destination – Create a Destination* screen to continue.

____26. On the *New Destination – Change properties* sceen use the **Select** buttons to specify the queue manager and the SYSTEM.DEFAULT.LOCAL.QUEUE as the queue. Click **Finish** to continue.



This JMS configuration information process has been saved in a file named *.bindings* in the specified directory on the local machine. This file needs to transmitted to z/OS in binary mode to OMVS directory /u/teamxx/jndi.

____27. Open the *Command Prompt* icon on the desktop. The properites of this icon have been set to start in the directory that contains the .bindings file created by using MQ Explorer.

____28. Start an file transfer session to the assigned host system, e.g. ***ftp mpx1***

____29. Provide the assigned userid and password.

____30. Once the userid has been authenticated, use the change directory (cd) command to change the current location to directory /u/teamxx/jndi, e.g. ***cd /u/teamxx/jndi***

____31. The *.binding* file needs to be moved to OMVS without being converted to EBCDIC. This is done by entering the ftp subcommand *bin*, e.g **bin**

____32. Start the transfer of the binding file by using the ftp subcommand *mput*, e.g. ***mput .bindings***

____33. Once the transfer is complete terminate the FTP session by enter the ***quit*** command

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

c:\z\jndi>ftp mpx1
Connected to mpx1.
220-FTPSERVE IBM FTP CS V1R13 at MPX1, 13:18:08 on 2015-06-18.
220 Connection will close if idle for more than 30 minutes.
User (mpx1:(none)): teamxx
331 Send password please.
Password:
230 TEAMXX is logged on.  Working directory is "TEAMXX.".
ftp> cd /u/teamxx/jndi
250 HFS directory /u/teamxx/jndi is the current working directory
ftp> bin
200 Representation type is Image
ftp> mput .bindings
mput .bindings? y
200 Port request OK.
125 Storing data set /u/teamxx/jndi/.bindings
250 Transfer completed successfully.
ftp: 18734 bytes sent in 0.24Seconds 79.05Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.

c:\z\jndi>
```

This completes this part of the exercise.

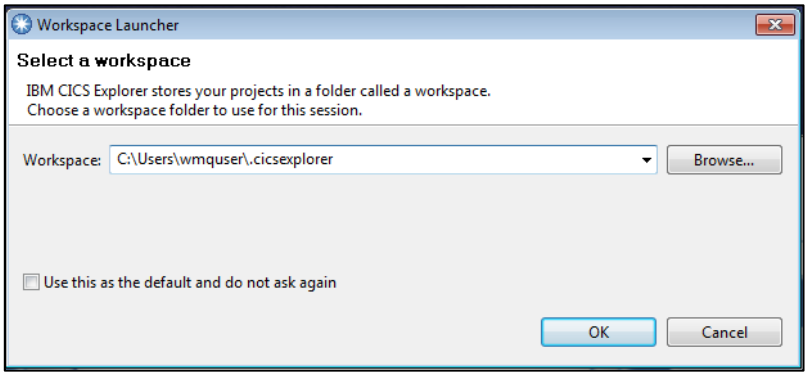
In this part MQ Explorer was used to configure JNDI information in a file system JNDI namespace. The JNDI file was moved to an OMVS directory that is accessible from CICS.

Part 3 - CICS Explorer and JMS Applications

CICS Explorer will be used in the part of the exercise to review the Java source and to make a few changes to the JMS web application. These changes are required so multiple instances of the JMS web application can be running concurrently in the same CICS region. The JMS web application will then be deployed and installed in CICS after these changes are made. The JMS enable CICS application was written so that one instance of the program in CICS could support multiple teams concurrently.

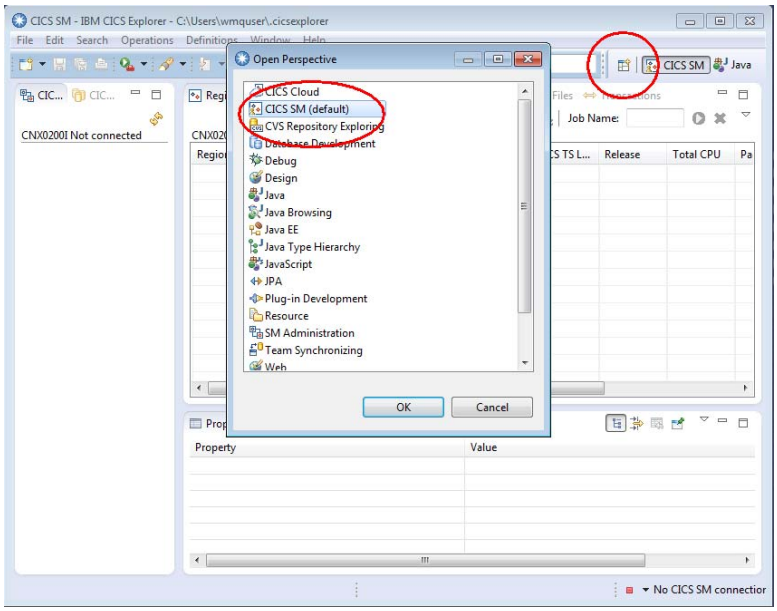
___ *1.* Start *CICS Explorer* by clicking on the *CICS Explorer* icon on the desktop.

___ *2.* If the *Workspace Launcher – Select a workspace* window is displayed let the *Workspace* default to C***:\Users\wmquser\.cicsplorer*** and press **OK** to continue.



Tech-Tip: Adding additional functionality to an Eclipse instance is done by installing plug-ins. Most Eclipse tooling for IBM products allow for the installation of the plug-ins of other IBM products such as CICS Explorer, z/OS Explorer, etc.

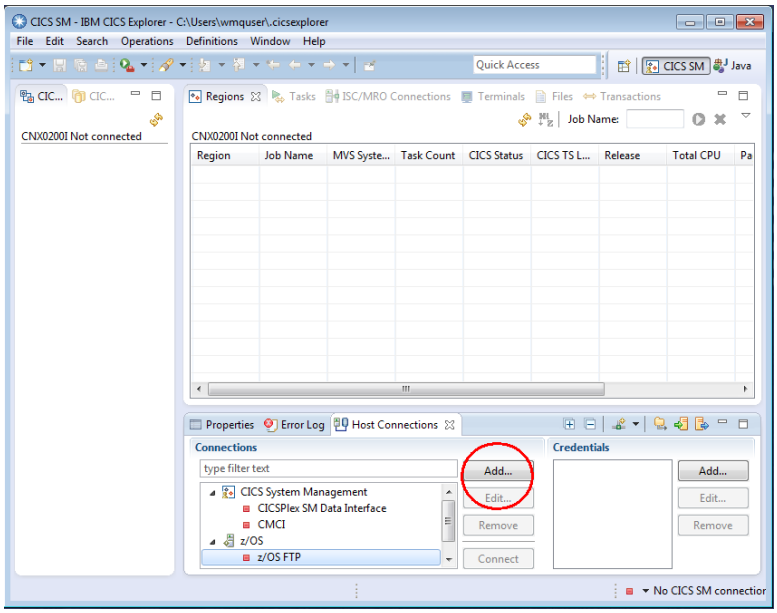
___ *3.* If the initial perspective is not *CICS SM* (see below), use the *Open Perspective* icon beside the perspective name and select *CICS SM(default)* from the *Open Perspective* list and click **OK** to continue.



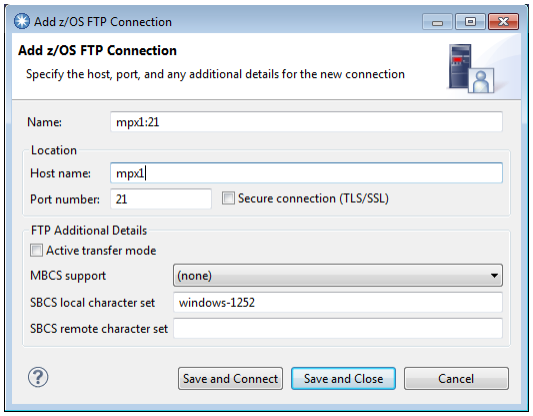
Tech-Tip: Eclipse based development tools like CICS Explorer provide a graphical interface consisting of a perspective which contains a tool bar and multiple views. Each perspective is tailored for the current role of the user. A web services developer would see a perspective and views tailored for developing web services, a Java developer would see a perspective layout views tailored for Java development and so on. This exercise uses the layout tailored for Java users. The current perspective can be identified by the title area of the perspective window.

A view is an area in the window dedicated to providing a specific tool or function within a perspective. For example in the perspective window above, *CICSPlex Explorer* and *CICSPlex Repositories* are views that happen to use the same area of the window for displaying information. To select which view is to be displayed simply click on the tab area where view name is displayed. There can be multiple areas in a window used displaying the contents of multiple views and these are commonly called stacked *views.* Another example on this perspective of a stacked view is *Properties, Error Log* and *Host Connections* at the bottom of the perspective window. In any stacked view the contents of each view can be displayed by clicking on the view tab (the name of the view).

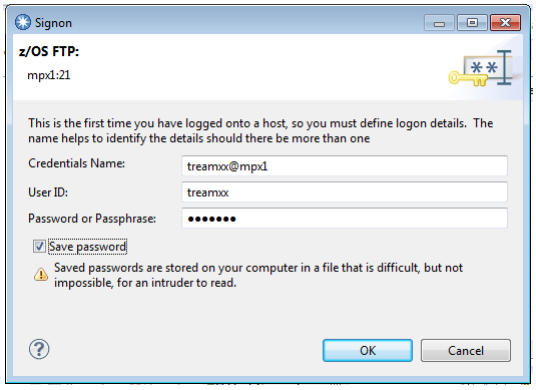
___ 4. CICS Explorer will eventually be used to deploy an application bundle to CICS for subsequent testing. This will require an FTP connection from CICS Explorer to the assigned z/OS system. To configure this connection select the *Host Connections* tab in the lower view and click on the *z/OS FTP* selection under the *z/OS* the Connections pane. Click on the **Add** button to continue.



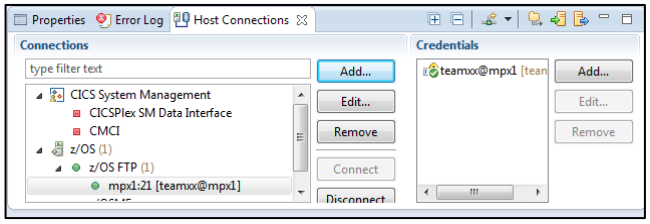
___ 5. On the *Add z/OS FTP Connection* screen enter the TCP/IP host name of the assigned z/OS in the area beside *Host name* and click **Save and Connect** to continue.



___ 6. On the *Signon – z/OS FTP* screen enter your assigned *User ID* and *Password* (saving the password is not an issue for our purposes so go ahead and save the password) in the areas beside *Userid ID* and *Password or Passphrase*. Click **OK** to continue.

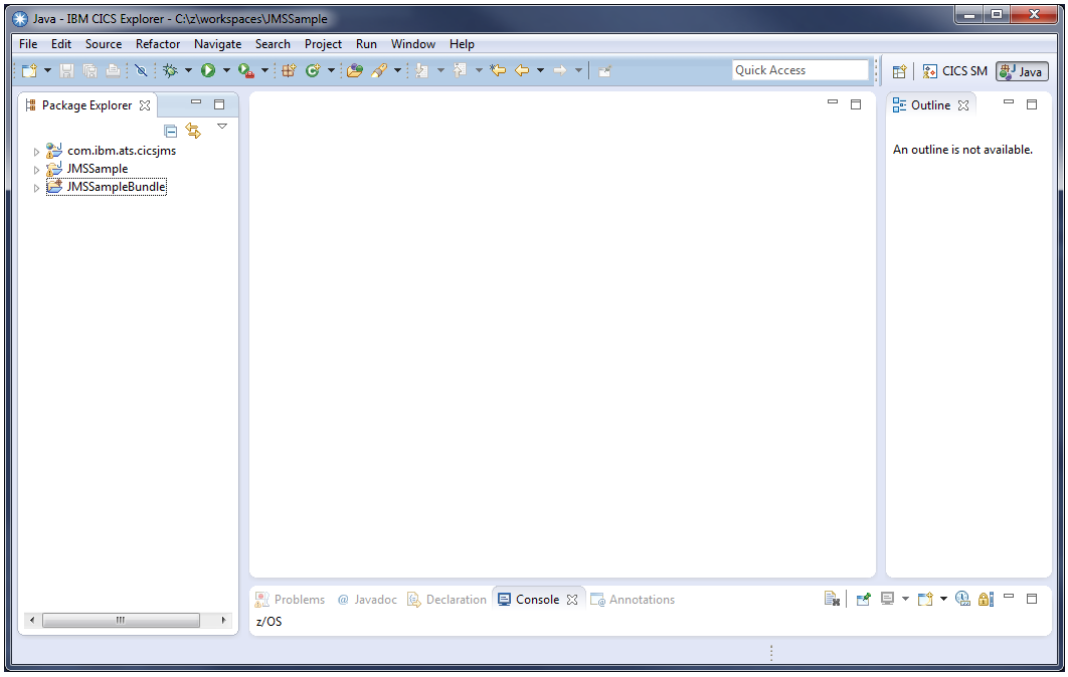


___ 7. There should be a green dot beside connection name (e.g. *mpx1:21)* showing that an FTP connection has been established (see below).



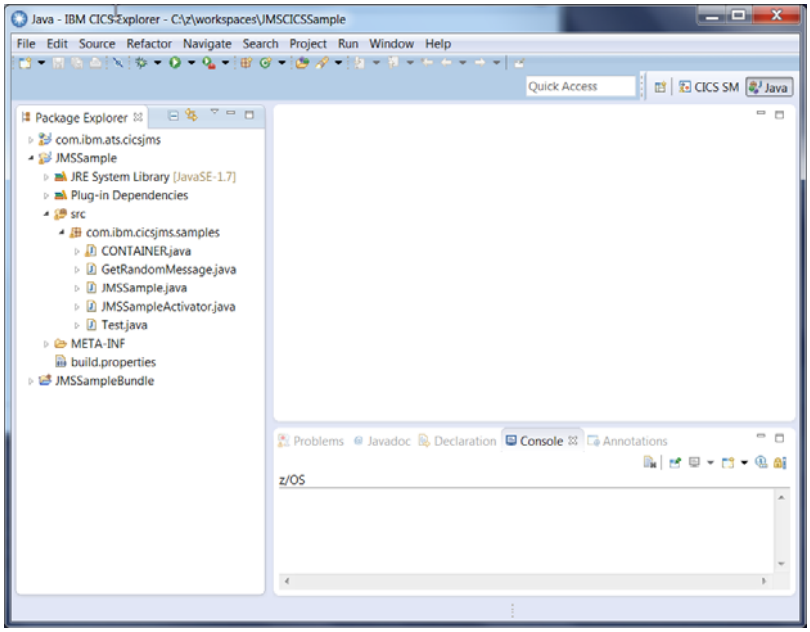
___ 8. Now switch to the *Java* perspective (see Step 2).

___ 9. In the *Package Explorer* view there should be several Java projects (see below).

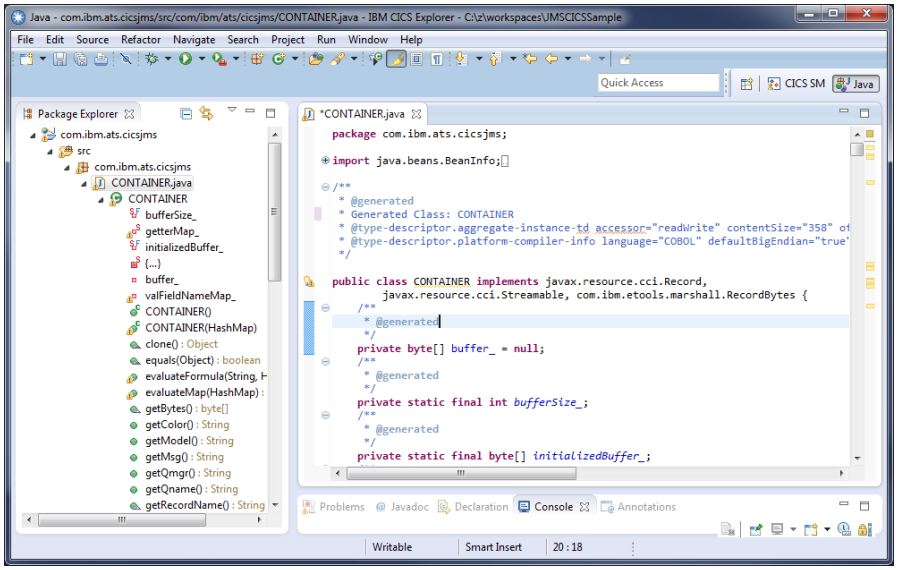


___ 10. The project *JMSSample* contains the CICS JMS enabled application while project *com.ibm.ats.cicsjms* contains the JMS web application. They have to be in separate projects because support for CICS JMS and Liberty are not supported in the same CICS JVM Server resource.

___ 11. Let's begin by exploring the *JMSSample* project. Expand this project and then expand the folder *src* and then the Java package *com.ibm.ats.cicsjms.samples* in order to display the Java classes in this project.



___ 12. Java class *CONTAINER.java* is a Java data bean created from a CICS COPYBOOK using Rational Application Developer or RAD. RAD wizards generated the getter and setter methods for all of the fields in the COPYBOOK. There are other tools to perform the same function but using RAD was simpler for the purposes of this exercise.



___ 13. Java class *JMSSampleActivator.java* is invoked prior to the execution of the main Java class in order to register the MQ object factories used by the main Java class with the JNDI provider.

___ 14. Java class *JMSSample.java* is the main CICS Java class that interacts with JMS. Below is the initial code that shows:

- The opening of the CICS channel to obtain the request container
- The use of JCICS class TSQ to write information from the container to a temporary storage queue (useful for debugging)
- The initial setup of the JNDI services (identifying the directory where the .bindings file can be located)

JMSSample.java

```
public static void main(com.ibm.cics.server.CommAreaHolder cah)
    {
        com.ibm.cicsjms.samples.CONTAINER workContainer = new CONTAINER();
        Channel channel = null;                                  // CICS channel
        Container requestContainer = null;                       // CICS request container
        Container responseContainer = null;                      // CICS response container

        Task task = Task.getTask();                              // Obtain CICS task object
        channel = task.getCurrentChannel();                      // EXEC CICS ASSIGN CHANNEL

        try {

            requestContainer = channel.getContainer("JMSSAMPRequest");  // EXEC CICS GET CONTAINER

            // Create JMS objects
            javax.jms.Connection connection = null;
            javax.jms.Session session = null;
            javax.jms.Destination destination = null;
            javax.jms.MessageProducer producer = null;
            javax.jms.MessageConsumer consumer = null;
            javax.jms.TextMessage message = null;

            // Copy request container to local object access contents
            workContainer.setBytes(requestContainer.get());
            String queueManager = workContainer.getQmgr().trim();
            String queueName = workContainer.getQname().trim();
            String color =  workContainer.getColor().trim();
            String model = workContainer.getModel().trim();

            // Create TSQ object
            TSQ tsqQ = new TSQ();
            tsqQ.setName("JMSSAMPLE");

            // Write container to temporary storage (useful for debugging)
            try {
                tsqQ.writeString(task.getUSERID() +
                                 workContainer.getRequest() +
                                 workContainer.getUserid() +
                                 workContainer.getQname() +
                                 workContainer.getQmgr() +
                                 workContainer.getColor() +
                                 workContainer.getModel() );
                tsqQ.writeString("file:///u/" + workContainer.getUserid().trim() + "/jndi/");
            } catch (Exception e) {
                workContainer.setMsg(e.getMessage());
            }

            //Create the JNDI initial context environment
            Hashtable<String, String> environment = new Hashtable<>();
```

___ 15. The next section of *JMSample.java* shows:

- The establishment of initial context into the JNDI service
- The use of the initial context to obtain a queue manager connection factory
- The use of the initial context to obtain a queue
- The establishment of a connection to the queue manager
- The creation of a session based on desired session properties
- The starting of the connection

```
*JMSSample.java

        // Instantiate the initial context
        Context context = new InitialContext(environment);

        // Lookup the connection factory using the value of variable "queueManager"  (Qmgr in COMMAREA)
        // This name must be defined in the .bindings file in the directory specified by PROVIDER_URL
        ConnectionFactory cf = (ConnectionFactory) context.lookup(queueManager);

         // Lookup the destination (queue) using value of variable "queueName" (QName in COMMAREA)
         // This name must be defined in the .bindings file in the directory specified by PROVIDER_URL
         destination = (Destination) context.lookup(queueName);

         //Connect to MQ and create a session that will allow messages to be processed under the CICS UOW.
         connection = cf.createConnection();

         //Create a session using the established connection
        session = connection.createSession(true, Session.SESSION_TRANSACTED);

        //Start the connection
        connection.start();
```

___ 16. The next section of *JMSample.java* shows the code to PUT a message on a queue either with or without message properties.

- A JMS text message object is created with a random quote
- If present, message properties are added to the message object
- A JMS sender object is created from the target queue object
- The message is placed on the queue
- If the put was successful, the task performs a EXEC CICS SYNCPOINT
- Otherwise, the error message is return to the calling application in the commarea
- Finally the connection is closed

```
Java - JMSSample/src/com/ibm/cicsjms/samples/JMSSample.java - IBM CICS Explorer - C:\z\workspaces\JMSCICSSample
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access | CICS SM | Java
JMSSample.java

        if (workContainer.getRequest().toLowerCase().equals("put")) {          // PUT message

            message = session.createTextMessage(new GetRandomMessage().getRandomMessage());

            // Set message properties
            if (!color.equals("none")) {
                message.setStringProperty("Color",color);
            }
            if (!model.equals("none")) {
                message.setStringProperty("Model",model);
            }

            // Create the JMS message producer (sender) object
            producer =   session.createProducer(destination);
            // Send the message using the JMS Producer
            producer.send(message);

            // Commit the message
            try {
                task.commit();
            } catch (Exception e){
                workContainer.setMsg(e.getMessage());
            }

            // Copy the message to the COMMAREA
            workContainer.setMsg(message.getText());

Writable | Smart Insert | 29 : 24
```

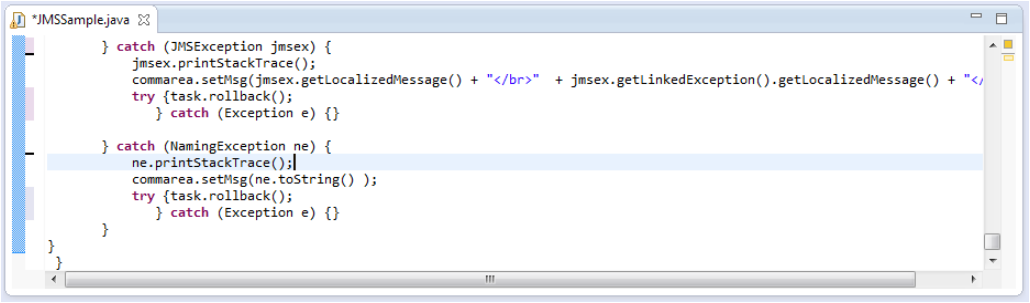
___ 17. This section of *JMSample.java* shows the code to GET a message from a queue either with or without message properties specified.

- A message properties selector string is created based on the presence of message properties in the request
- A JMS message consumer is created either with or with a message properties selector string.
- A get request for a message is performed using the JMS consumer
- The results are analyzed for the presence of message properties and the message properties fields are set in the container.
- The returned message is placed in a response container
- The connection is closed.

JMSSample.java

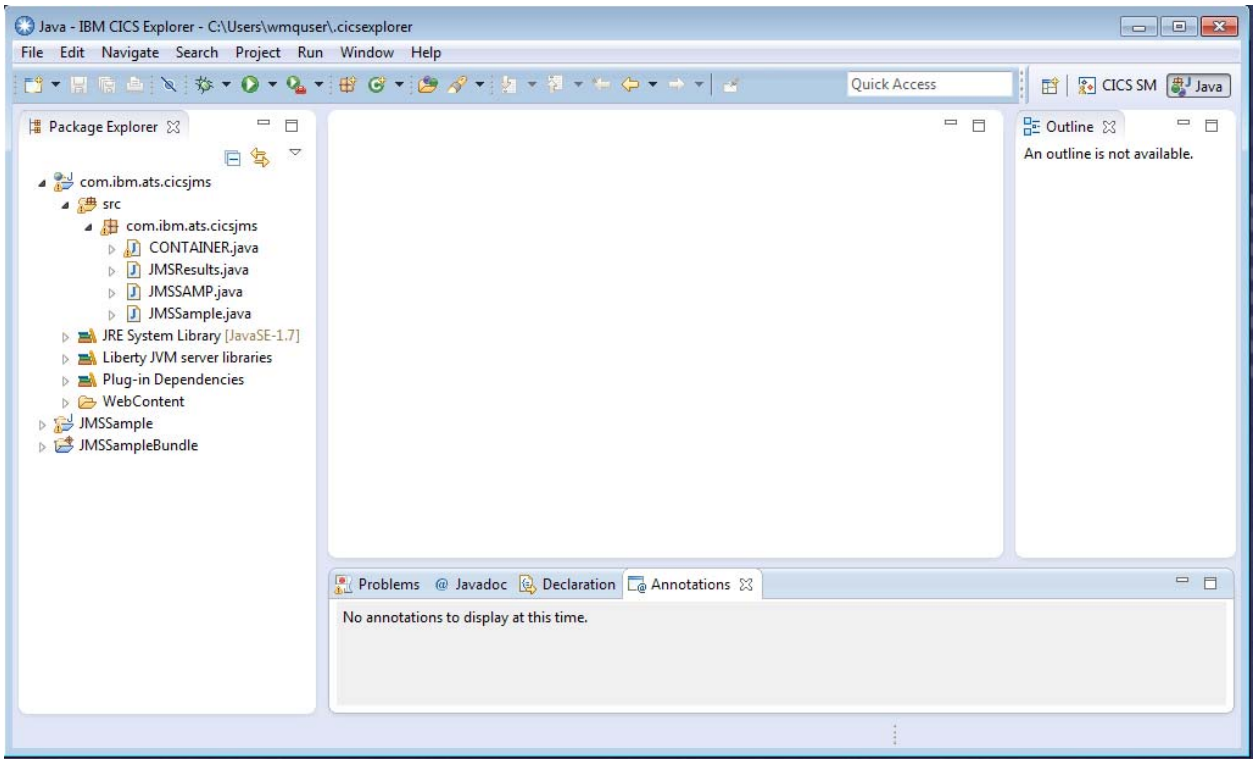
```
        } else {   // GET message
            String selector = null;

            // Set the message selector string
            if (color.equals("none")) {
                if (!model.equals("none")) {
                    selector = "Model = '" + model + "'";
                }
            } else {
                selector = "Color = '" + color + "'";
                if (!model.equals("none")) {
                    selector = selector + " AND Model = '" + model + "'";
                }
            }

            // If message properties are specified, create the JMS Message consumer with a message properties selector string
            if (selector == null) {
                consumer =   session.createConsumer(destination);
            } else {
                consumer =   session.createConsumer(destination,selector);
            }

            // Receive the message
            message = (TextMessage) consumer.receive(1500);

            // Set the message properties COMMAREA fields
            if (message == null) {
                workContainer.setMsg("No matching message available");;
            } else    {
               workContainer.setMsg(message.getText());
            if (message.getStringProperty("Color") == null ) {
               workContainer.setColor(" ");
            } else {
               workContainer.setColor(message.getStringProperty("Color"));
            }

            if (message.getStringProperty("Model") == null ) {
              workContainer.setModel(" ");
            } else {
              workContainer.setModel(message.getStringProperty("Model"));
            }

            // Copy the retrieved messages to the COMMAREA
            workContainer.setMsg(message.getText());

            // Close the connection
            responseContainer = channel.createContainer("JMSSAMPResponse");
            responseContainer.put(workContainer.getBytes());              // EXEC CICS PUT CONTAINER
            connection.close();

             }
          }
```

___ 18. Finally exceptions are handled.



___ 19. No changes are required to *JMSSample* project for this exercise.

___ 20. The JMS web application can be found in project *com.ibm.ats.cicsjms*. Expand this project and then expand the folder *src* and then the Java package *com.ibm.ats.cicsjms* in order to display the Java classes in this project.



___ 21. Below is a section of code for Java class *JMSResults.java*. This code obtains the end user's selection from the web browser (entered on the JSP *JSPSample.jsp*) and then populates the CICS request container (Java object *CONTAINER*). The *invoke method* on Java class *com.ibm.ats.cicsjms.JMSSAMP* performs the CICS link to the CICS JMS application. Upon return from JMSSAMP the contents of the response container are place in the HTTP request for display by JSP *JMSResults.jsp*.

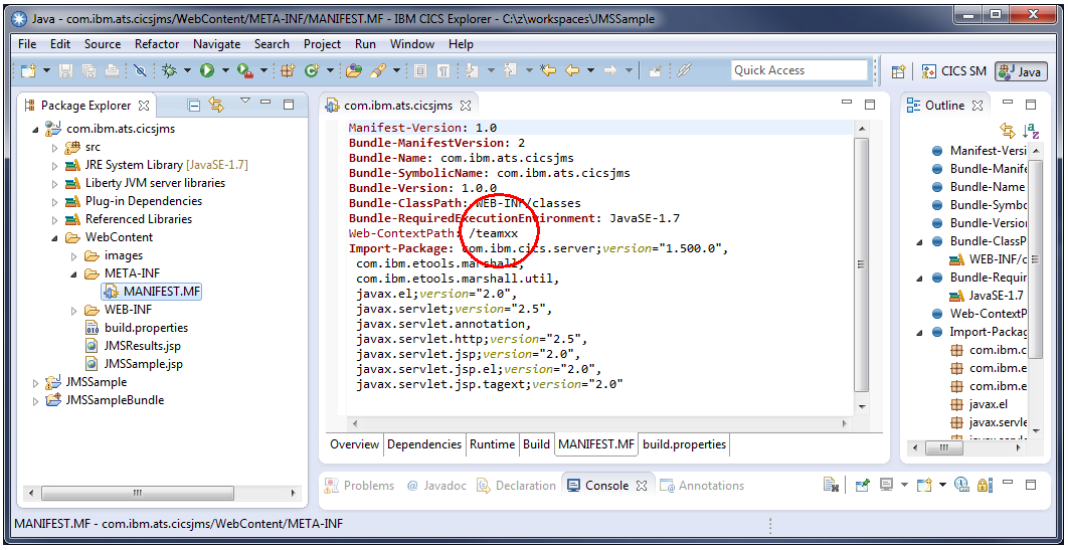
JMSResults.java

```
        CONTAINER container = new CONTAINER();

        container.setQmgr(request.getParameter("queueManager"));
        container.setQname(request.getParameter("queue"));
        container.setColor(request.getParameter("color"));
        container.setModel(request.getParameter("model"));
        container.setUserid(request.getParameter("userid"));

        request.setAttribute("errorMessage", "  ");
        request.setAttribute("message", "  ");
        request.setAttribute("color", " ");
        request.setAttribute("model", " ");

        if (request.getParameter("option").equals("put")) {
            container.setRequest("put");
        } else {
            container.setRequest("get");
        }

        container = com.ibm.ats.cicsjms.JMSSAMP.invoke(container);

        if (request.getParameter("option").equals("put")) {
            request.setAttribute("queueManager", container.getQmgr());
            request.setAttribute("queue",container.getQname());
            request.setAttribute("message", container.getMsg());
            request.setAttribute("color", request.getParameter("color"));
            request.setAttribute("model", request.getParameter("model"));
            request.setAttribute("errorMessage", "Message successfully sent");
        } else {
            if (container.getMsg() == null) {
                request.setAttribute("message", "No message received");
                ;
            } else {
                request.setAttribute("message", container.getMsg());
                if (container.getColor() == null) {
                    request.setAttribute("color", " ");
                } else {
                    request.setAttribute("color", container.getColor());
                }
                if (container.getModel() == null) {
                    request.setAttribute("model", " ");
                } else {
                    request.setAttribute("model", container.getModel());
                }
            }
```

___ 22. Below shows the contents of Java class *JMSSAMP.java*. This is the code that uses the CICS JCICS *link* method of a CICS JCICS *Program* object in order to perform a Java equivalent of an EXEC CICS LINK.

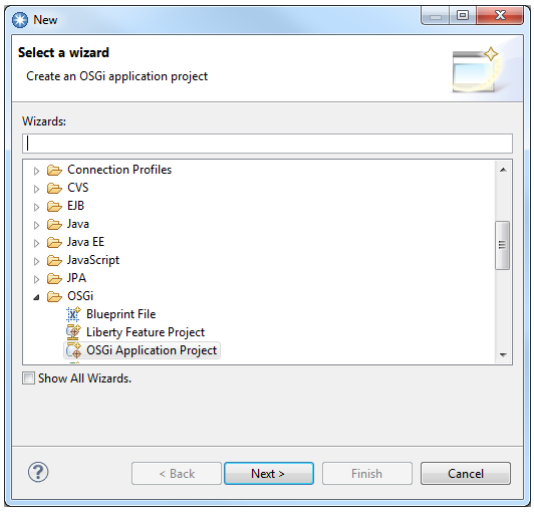
```
JMSSAMP.java
package com.ibm.ats.cicsjms;

import com.ibm.ats.cicsjms.CONTAINER;

public class JMSSAMP {
    public static com.ibm.ats.cicsjms.CONTAINER invoke(CONTAINER container) {

    // Instantiate an instance of a JCICS program class and set the CICS program name
    com.ibm.cics.server.Program progName = new Program();
    progName.setName("JMSSAMP");
    // Instantiate an instance of a JCICS task class
    com.ibm.cics.server.Task task = com.ibm.cics.server.Task.getTask();

    try {

        //  Use the JCICS program link method to invoke program and pass the commarea
        // progName.link(commarea.getBytes());
        // Instantiate an instance of a JCICS channel class and an instance of a JCICS container class
        Channel channel = task.createChannel("JMSSAMP-Channel");
        com.ibm.cics.server.Container requestContainer = channel.createContainer("JMSSAMPRequest");
        // Copy contents of the CICS Java Data Binding request container to the JCICS request container instance
        requestContainer.put(container.getBytes());
        //  Use the JCICS program link method to invoke program and pass the channel with the request container
        progName.link(channel);
        //progName.link(commarea.getBytes());
        // Retrieve the response container from the channel
        com.ibm.cics.server.Container responseContainer = channel.getContainer("JMSSAMPResponse");
        // Copy contents of the JCICS response container to the CICS Java Data Binding response container instance
        container.setBytes(responseContainer.get());

    } catch (Exception e) {
        e.printStackTrace();
    }
    return container;
    }
}
```

___ 23. In order for each team to have a unique URI when testing, the JSP web application requires a couple of changes. Expand project *com.ibm.ats.cicsjms* and expand folder *WebContent* and then expand folder *META-INF* and then double click *MANIFEST-MF* to open the project's manifest file for editing. Go to the MANIFEST.MF tab and change the *Web-ContextPath* variable so it has your assigned team number, e.g. ***/teamxx*** (see below).



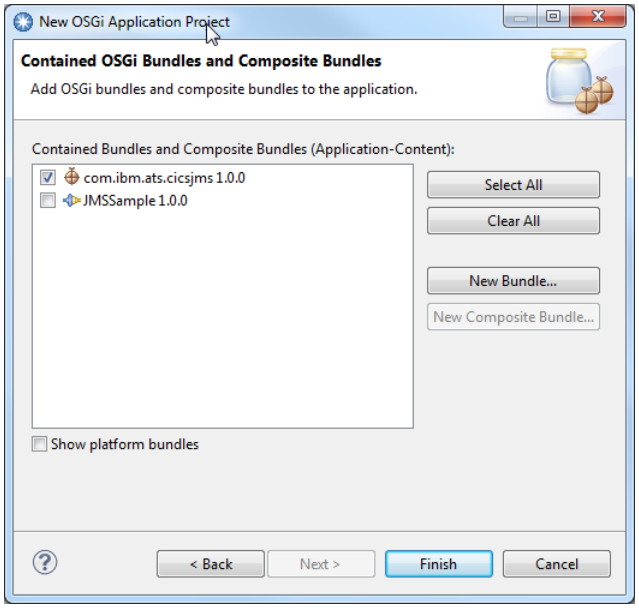
Tech-Tip: The asterisks in front of *com.ibm.ats.cicsjms* in the tab's title indicates that the contents this file have changed since it was opened. To save changes, either use the ***Ctrl-S*** key sequence or save the change when closing the open tab entry. *It is a good practice to save changes as they are made*.

___ *24.* The *com.ibm.ats.cicsjms* project now needs be included in an OSGi application project. Select **File** ->**New** -> **Other** and then expand the *OSGi* folder and then select *OSGi Application Project* wizard. Click **Next** to continue.

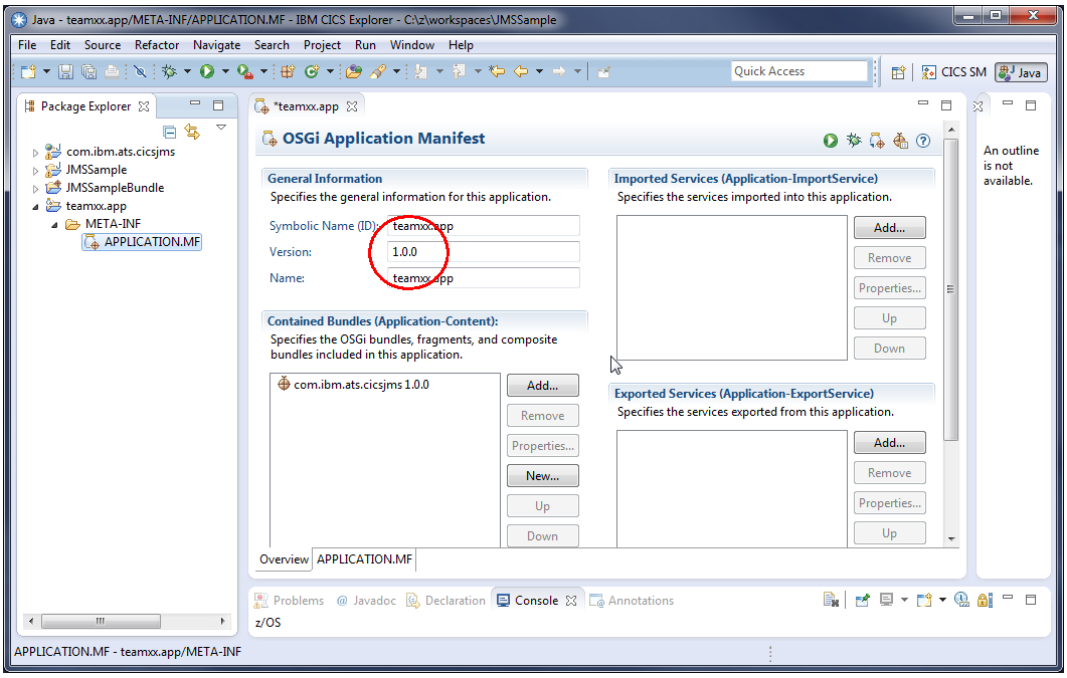


___ 25. On the *New OSGI Application Project* screen enter ***teamxx.app*** in the area beside *Project name* and press **Next** to continue.

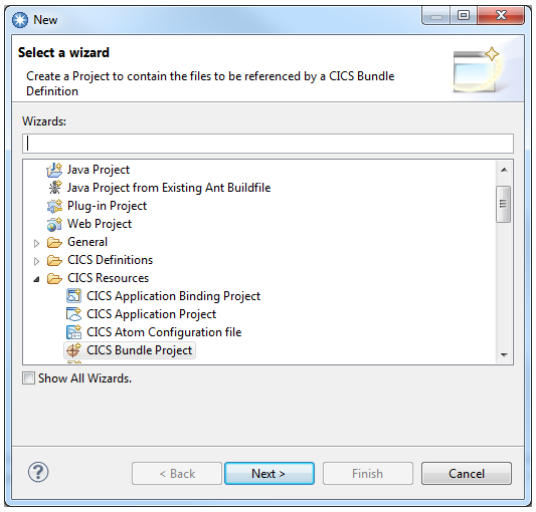
___ 26. On the *New OSGi Application Project-Contained OSGi Bundles and Composite Bundles* screen select the box beside *com.ibm.ats.cicsjms.1.0.0* and press **Finish**.



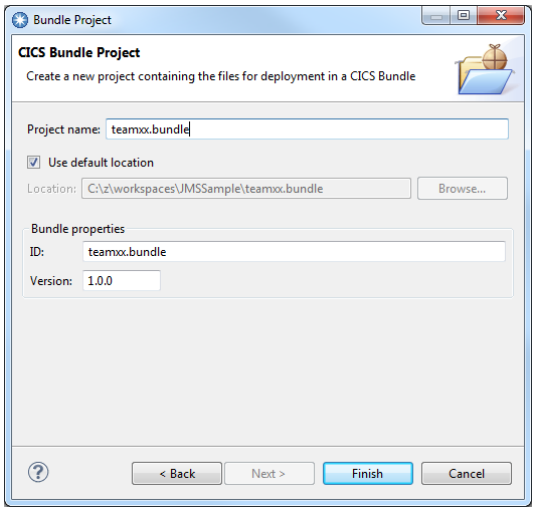
___ 27. Expand project *teamxx.app* and folder *META-INF* and double click *APPLICATION.MF* to open this file for editing. By default *.qualifier* was added to the application version. CICS does not support qualification so this string needs to be removed (including the period). Remove the qualification and save the change.



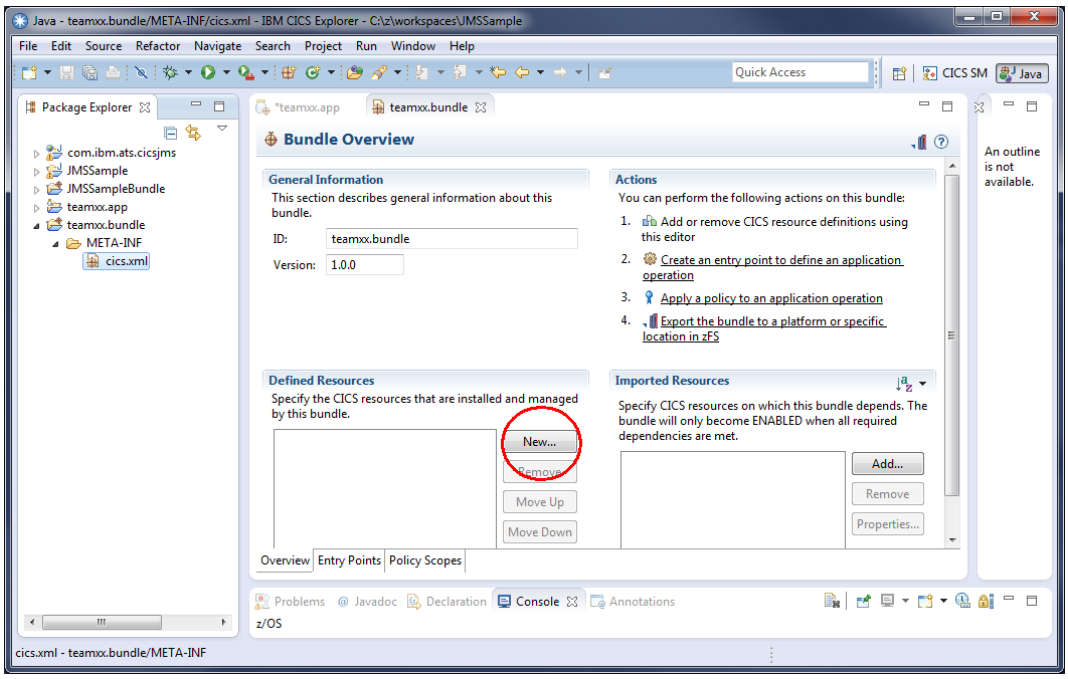
___ 28. The *teamxx.app* application project should be included in a CICS bundle project. Select **File** –>**New** –> **Other** and then expand the *CICS Resources* folder and then select the *CICS Bundle Project* wizard. Click **Next** to continue.



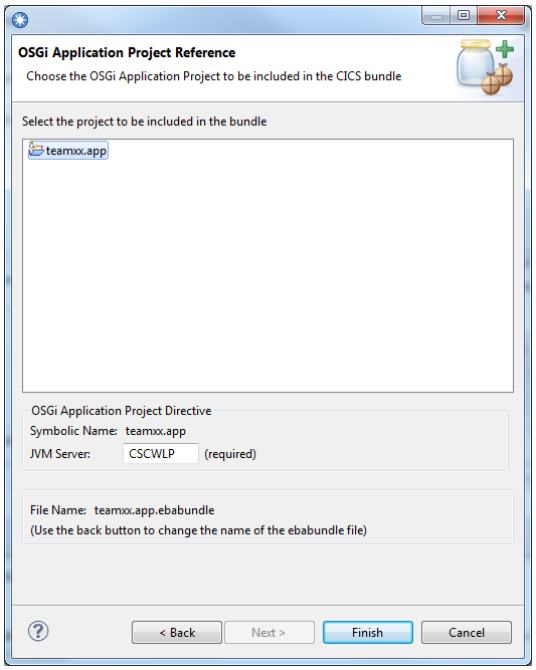
___ 29. On the *Bundle Project* screen enter ***teamxx.bundle*** in the area beside *Project name* and click **Finish** to continue.



___ 30. Expand project *teamxx.bundle* and folder *META-INF* and double click *cics.xml* to open this file with the *CICS Bundle Manifest Editor*. Click the **New** button in the *Defined Resources* area of the screen to start adding CICS related resources to this bundle.



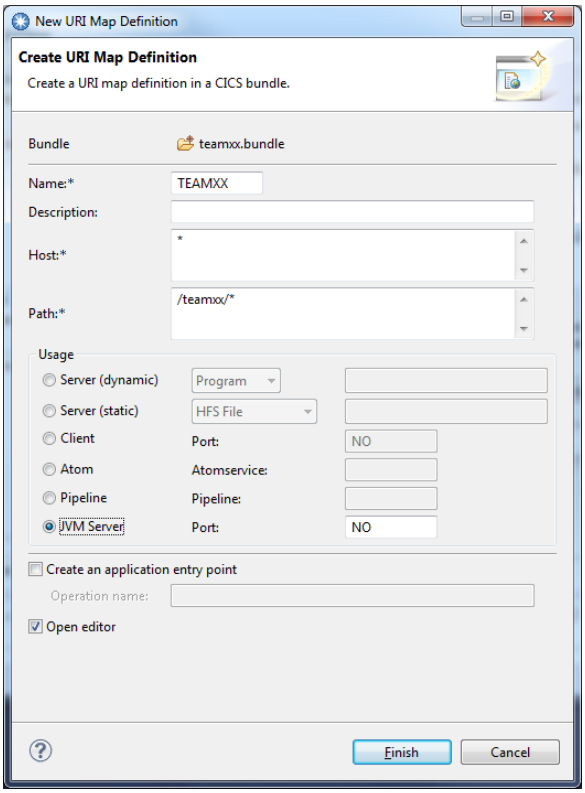
___ 31. Select the OSGI *Application Project Reference* options from the list to display the OSGi Bundle Project Reference screen. Select project *teamxx.app* and enter ***CSCWLP*** as the *JVM Server*. Click **Finish** to continue.



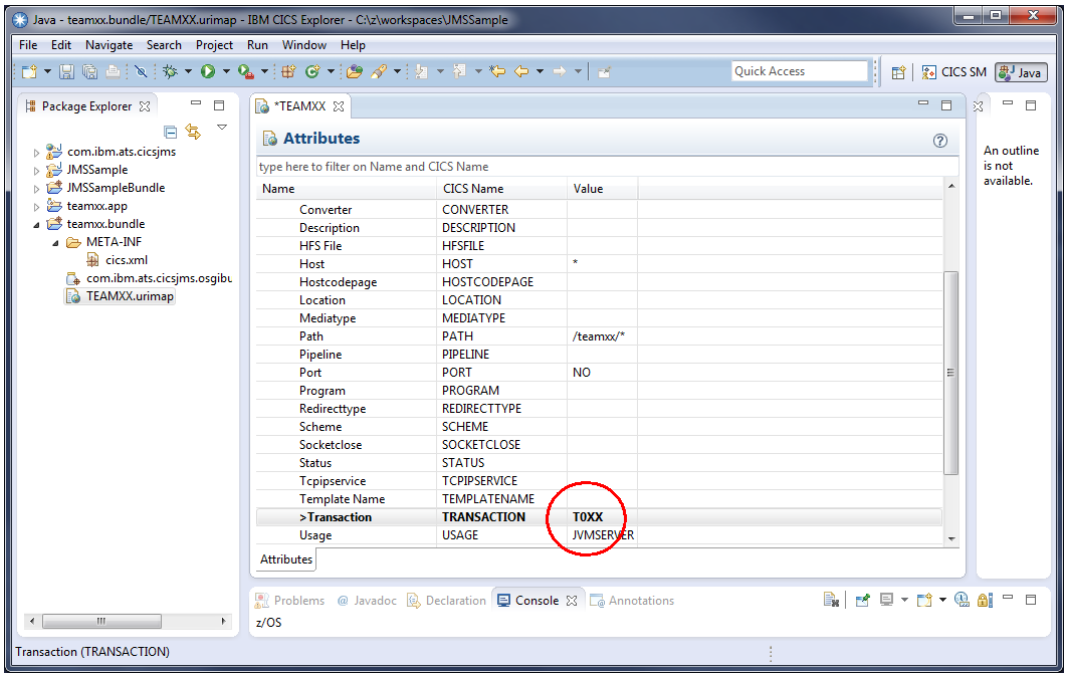
___ 32. The CICS transaction under which this application executes in CICS can be specified at this time by adding an *URI Map Definition* to the bundle. Begin by clicking the **New** button again in the *Define Resources* area of the screen and select the *URI MAP Definition* option from the list.

> **Tech-Tip:** By default transaction CJSA would be used for this application. The setting of an alternative transaction code allows the use of multiple EDF sessions for each team.

___ 33. This will open the *New URI MAP Definition* screen. Enter ***TEAMXX*** as the name of the definition and asterisk (*****) for the *Host* and ***/teamxx/**** for the *Path* of the URI. Select the radio button for the *JVM Server* under *Usage* and check the box beside *Open editor*. Click **Finish** to continue.



___ ***34.*** This action will open the editor for the URI MAP attributes. Scroll down the page and enter ***T0XX*** under column *Value* in the area beside row *Transaction*.



___ 35. The CICS transaction under which this application is executes in CICS can also be defined in the bundle. Click the **New** button again in the *Define Resources* area of the screen and select the *Transaction Definition* option from the list.

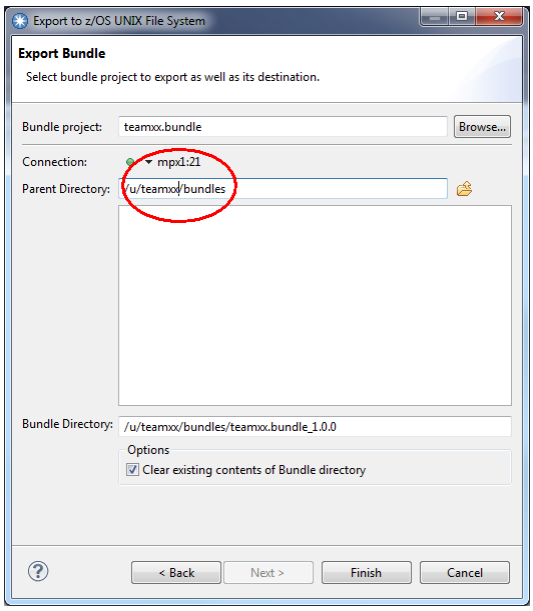
___ 36. This will open the *New Transaction Definition* screen. Enter ***T0XX*** as the name of the definition and ***DUMMY*** for the *Program Name*. Click **Finish** to continue.

> **Tech-Tip:** You can use a bogus program like DUMMY since the transaction will not be used in a START request. Program DUMMY has been defined to CICS but it does not exist. Since we specified T0XX as our runtime transaction, CICS will simply switch from using CJSA to this transaction context during execution. This is an easy way to allow us to use EDF just on single team's invocation of the target program and would be useful for other areas when having a unique transaction would be useful, e.g. security.

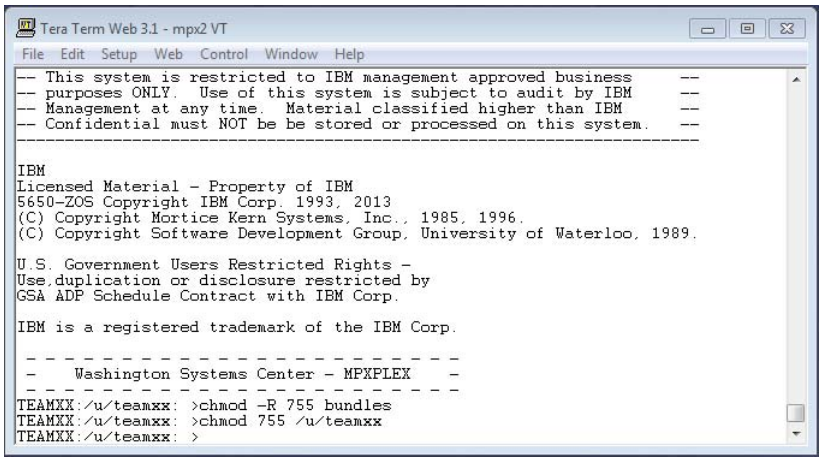
___ 37. After these changes have been made the application bundle *teamxx.bundle* needs to be deployed to an OMVS directory. Save all changes and select *teamxx.bundle* and right mouse button click and select *Export bundle project to z/OS UNIX File System*.

___ 38. On the *Export to z/OS UNIX File System – Export Bundle* screen select the radio button beside *Export to a specific location in the file system* and press **Next** to continue.

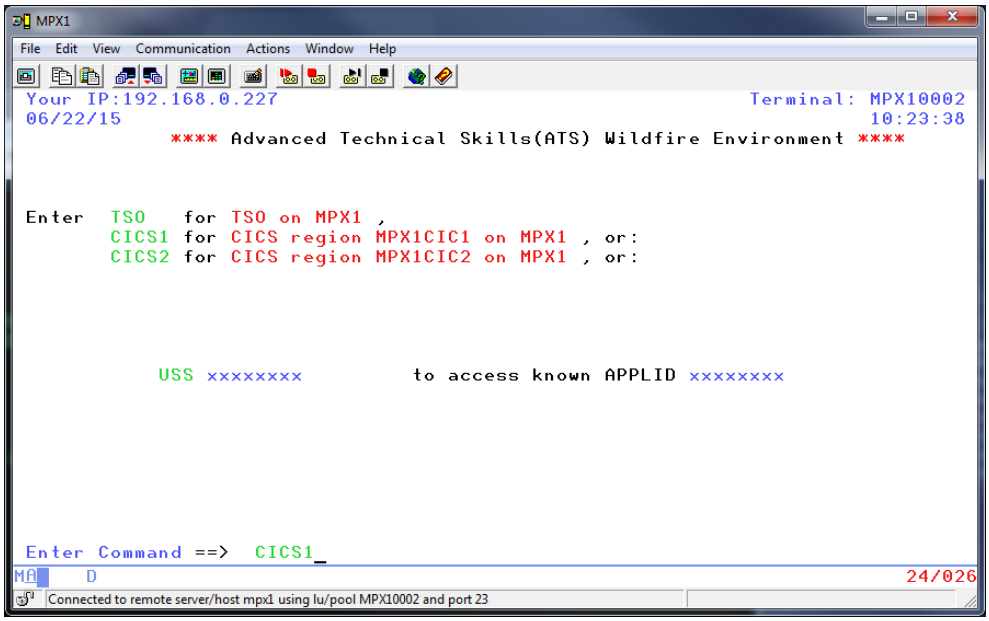
___ 39. On the next *Export to z/OS UNIX File System – Export Bundle* screen enter the directory name ***/u/teamxx/bundle*** in the area beside *Parent Directory* and press **Finish** to have the CICS bundle transferred to the target OMVS directory. Note that once the directory has been populated the box beside *Clear existing contents of Bundle directory* must be checked in order to replace any existing bundle.



___ 40. Start an OMVS session to the assigned z/OS system using the Tera Term icon on the desktop and enter commands ***chmod –R 755 /u/teamxx/bundles*** and ***chmod 755 /u/teamxx*** **in** the terminal session. This command gives the CICS region the necessary access to the bundle directory and its contents.

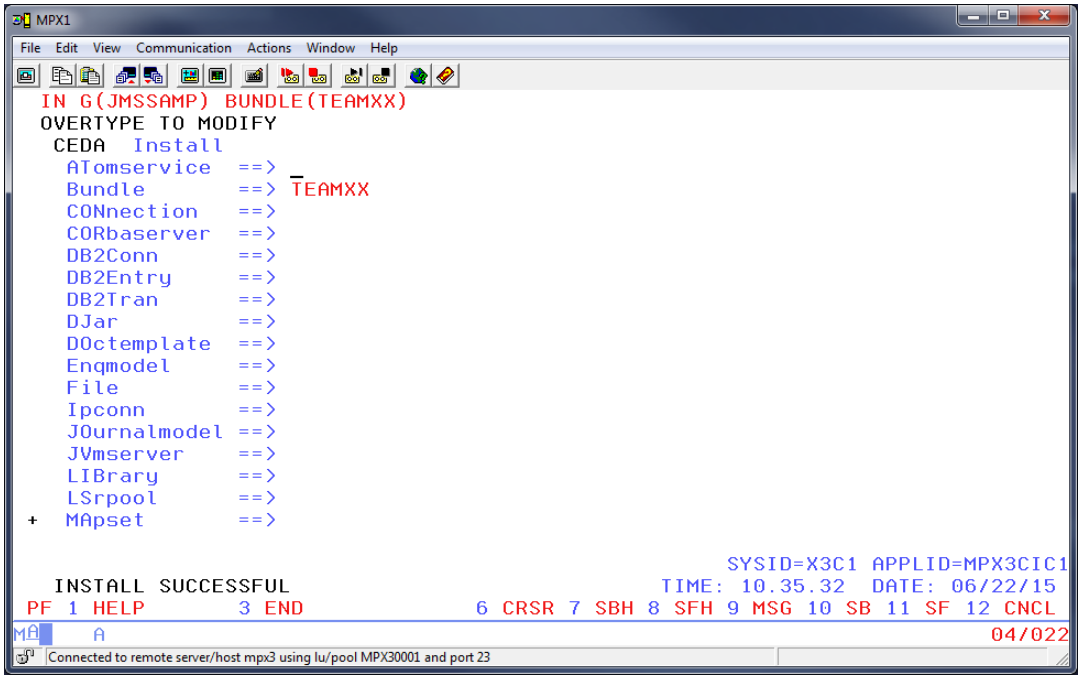


___ ***41.*** Start a 3270 session to your assigned z/OS LPAR and logon to CICS1.



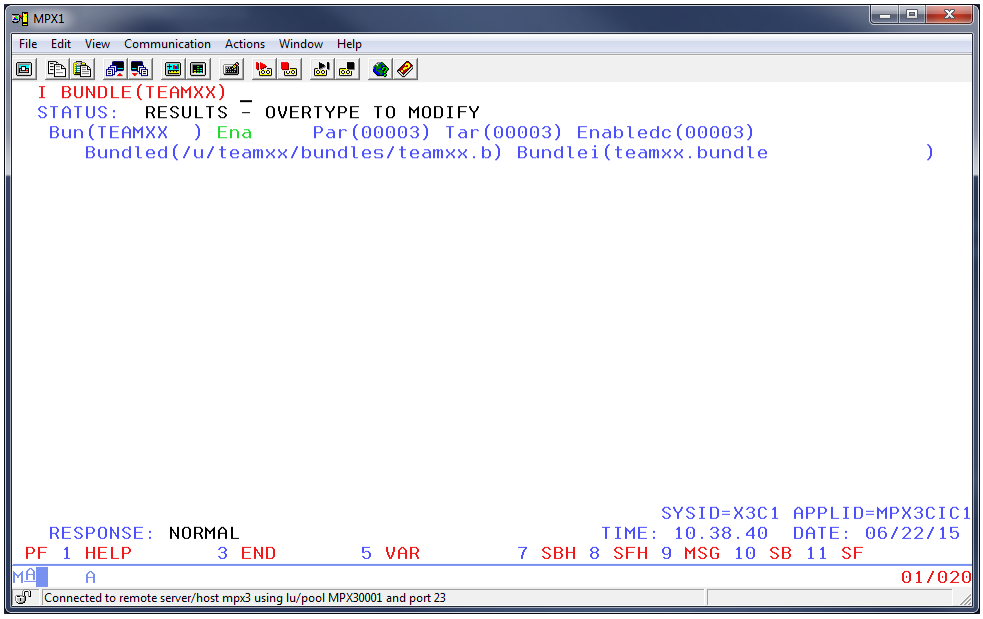
___ ***42.*** Clearing the WELCOME to CICS screen depends on the type of laptop and/or keyboard you are using. If your laptop has a *Pause* key use it to clear the screen. Newer laptops without a *Pause* key use the key sequence *Fn-P* to clear the screen. If none of these works, try a *Break* key.

___ ***43.*** Once the screen is clear enter CICS transaction ***CEDA IN G(JMSSAMP) BUNDLE(TEAMXX)*** to install the bundle containing the JMS web application.

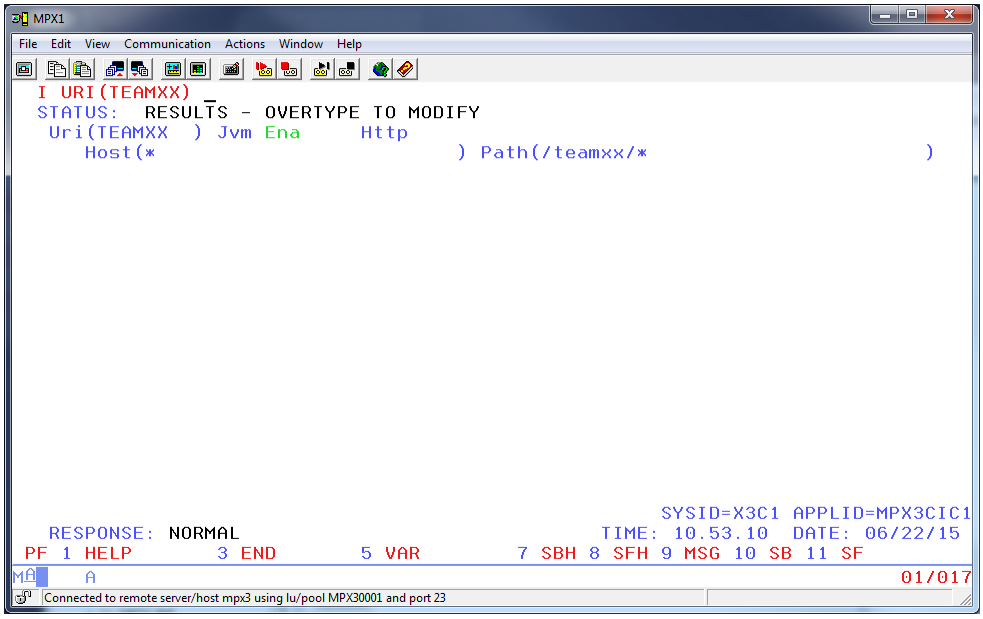


___ ***44.*** Use the F3 key to terminate the CEDA transaction and clear the screen again and use the CEMT transaction to verify the bundle, the transaction and URI map are installed.

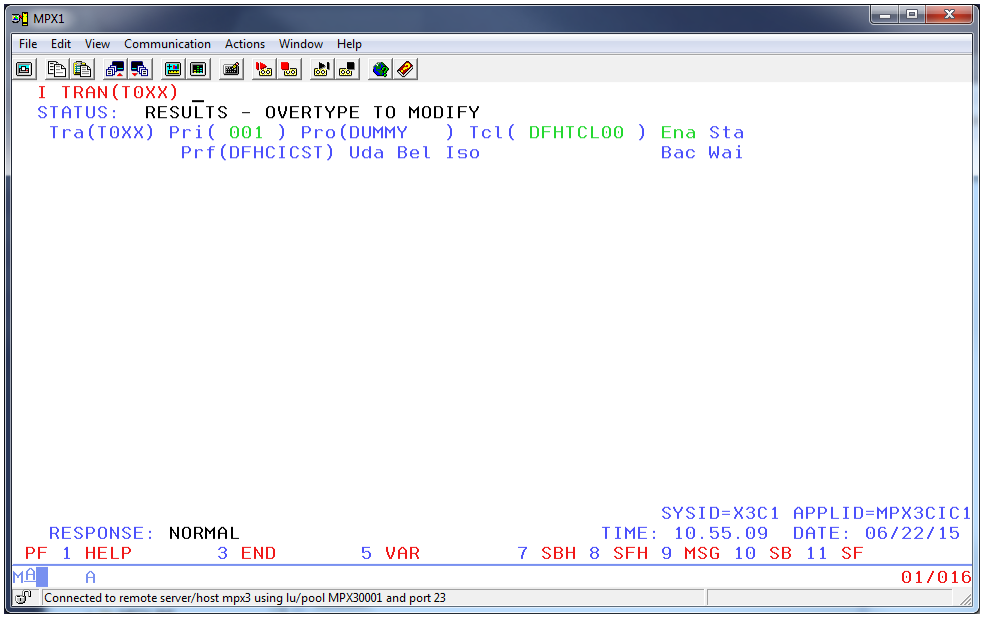
- CEMT I BUNDLE(TEAMXX)



- CEMT I URI(TEAMXX)



- CEMT I TRAN(T0XX)



This completes this part of the exercise.

In this part CICS Explorer was used to review the web and JMS application and then used to deploy the web application to CICS. A CICS 3270 terminal interface was used to install the web application. The installation could have just as easily been done using the SM interface in CICS Explorer, see CICS Bundle Management in the Appendix.

Part 4 - Testing the JMS Sample Application

Now that the JMS web application has been installed, it is time to use a web browser to test the JMS enabled CICS application.

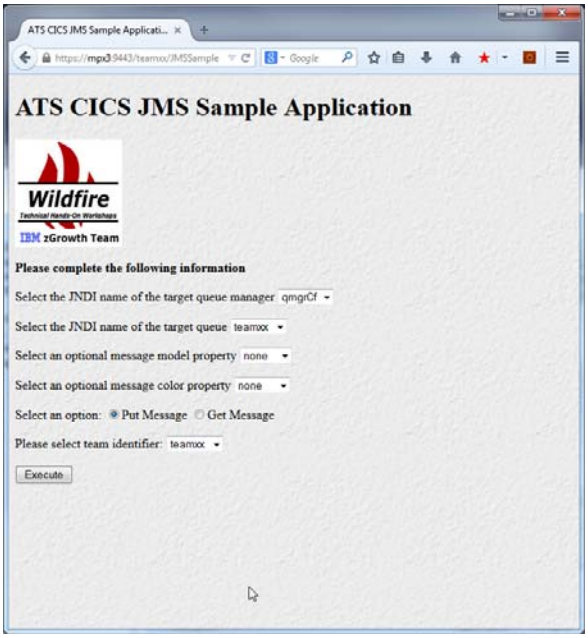
Testing will first use a web browser to execute the *ATS CICS JMS Sample Application* (both the JMS web application and the CICS JMS application) to put several messages onto the destination queue. MQ Explorer will be used to verify messages are written to the target queue with the desired message properties. Then CICS EDF transaction CEDX will be use to review the flow of application and the interactions of CICS resources.

Note that the CICS JMS application bundle was previously installed using the same technique just used to install the JMS web application.

___ 1. Access the JMS web application by opening a web browser and entering URL ***https://mpx1:9443/teamxx/JMSSample*.**

___ 2. If the web browser present a warning message about the connection not being trusted. Expand the *I Understand the Risks* section and click the **Add Exception** button to confirm a security exception for this site.

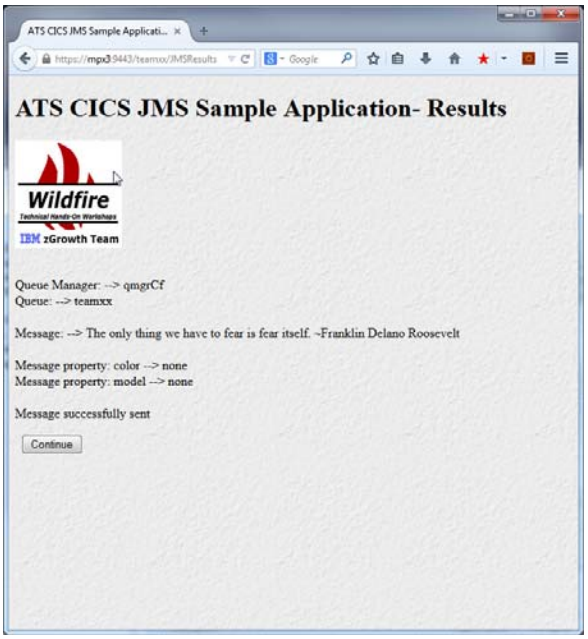
___ 3. Use the pull downs options to select a queue manager (only one selection is provided), a target queue (select either *default* or *teamxx*, these were configure back in Part 2 where you created the .bindings file) and a team identified, e.g. *teamxx*.



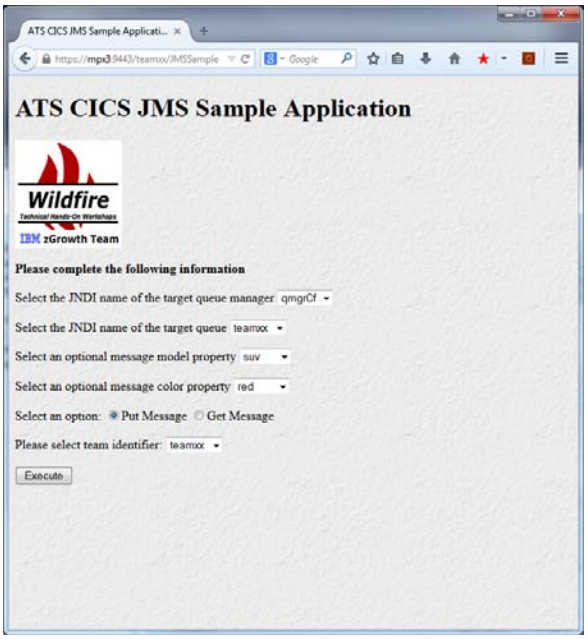
Tech-Tip: Multiple JNDI name options are provided for queue names. Use teamxx if you want to work independently of from other teams. The JNDI name defined in Part 2 should only occur in your *.bindings* file. The JNDI name of *default* references queue SYSTEM.DEFAULT.LOCAL.QUEUE and should appear in all .binding files. Use JNDI name default if you want to coordinate putting and getting messages with another team. Also, be very careful to not let the JNDI name of the queue or the team identifier default to teamxx. Otherwise the application will not access the intended queue or naming exceptions may occur.

___ 4. Optionally select a combination of properties. Valid model properties include *sedan, suv, truck* and *none* (for no model property) and valid color properties include *red, orange, yellow, green, blue* and *none* (for no color property).

___ 5. Select the radio button beside the *Put Message* and press the **Execute** button to continue. This should display the *ATS/CICS JMS Sample Result* screen indicating the message was successfully put on the selected queue. Press the **Continue** button to continue.

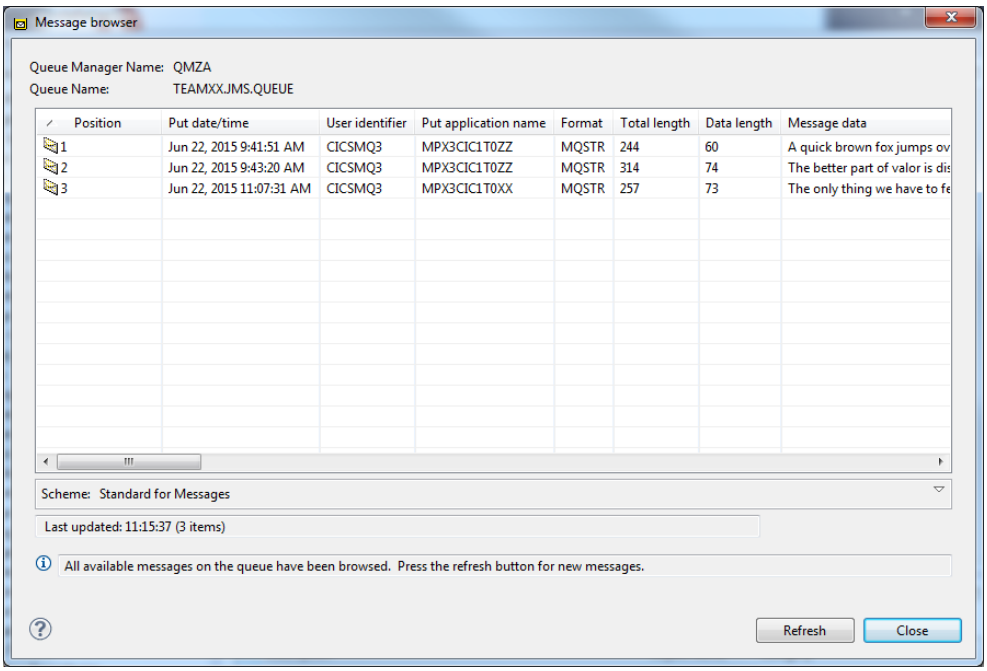


___ 6. Put several more messages on the same queue but this time select various combinations of message properties. For example set the model to *suv* and the color to *red* as shown below.

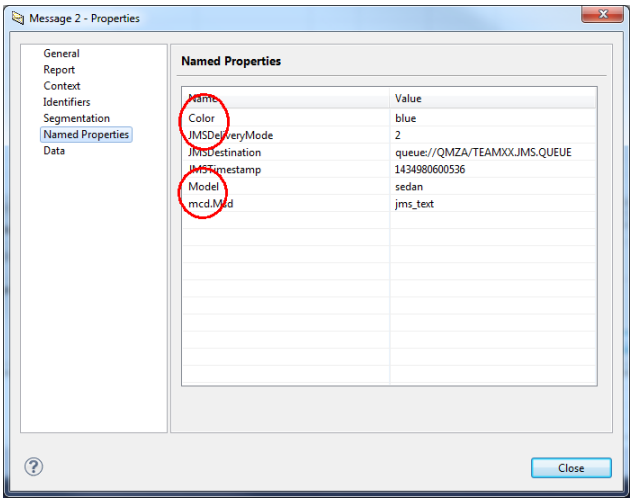


Now use the MQ Explorer to browse the contents on the queue and verify that the message properties were set as specified.

___ 7. If MQ Explorer is not active the start it by clicking on the desktop icon. Expand your queue manger and then expand its queues until your queue is displayed. Select your queue and right mouse button and select the *Browse Messages* option. This should display a list of the messages in your queue.



___ 8. Next start selecting the individual messages and right mouse button click. Select the *Named Properties* tab and confirm that the message properties for that message appear. What properties you see will depend entirely on which properties you selected when you put messages on the queue.



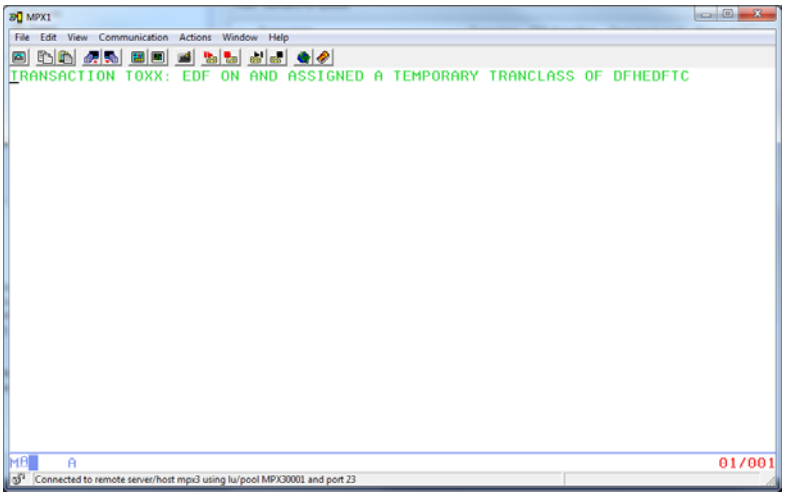
___ 9. Once you have a good set of messages with various combinations of message properties start selecting the **Get Message** button while selecting various message properties and verify the message retrieved has properties which match the criteria specified for message properties.

Tip; consider using the *MQ Explorer* to cross check the message properties displayed there versus the properties of the message retrieved. This might require keeping detail notes of which messages are in the queue and what properties they may or not have and considering multiple messages have the same combination of properties.

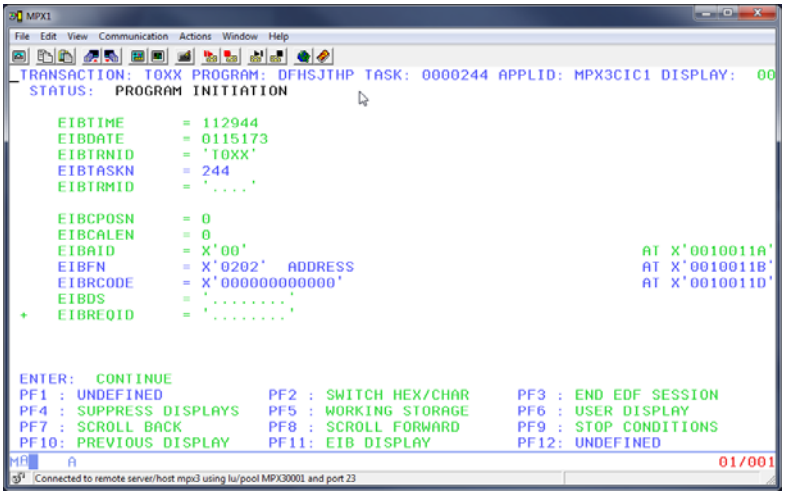
___ 10. In the web browser go back to the *ATS JMS Sample* Application page if not there already.

___ 11. If the CICS 3270 session used to install the web application bundle is not active, start a new 3270 session for your z/OS LPAR and start a session to ***CICS1***.

___ 12. Clear the CICS screen and enter transaction **CEDX T0XX**. This starts an EDF trace in the CICS terminal session any time transaction T0XX is started.

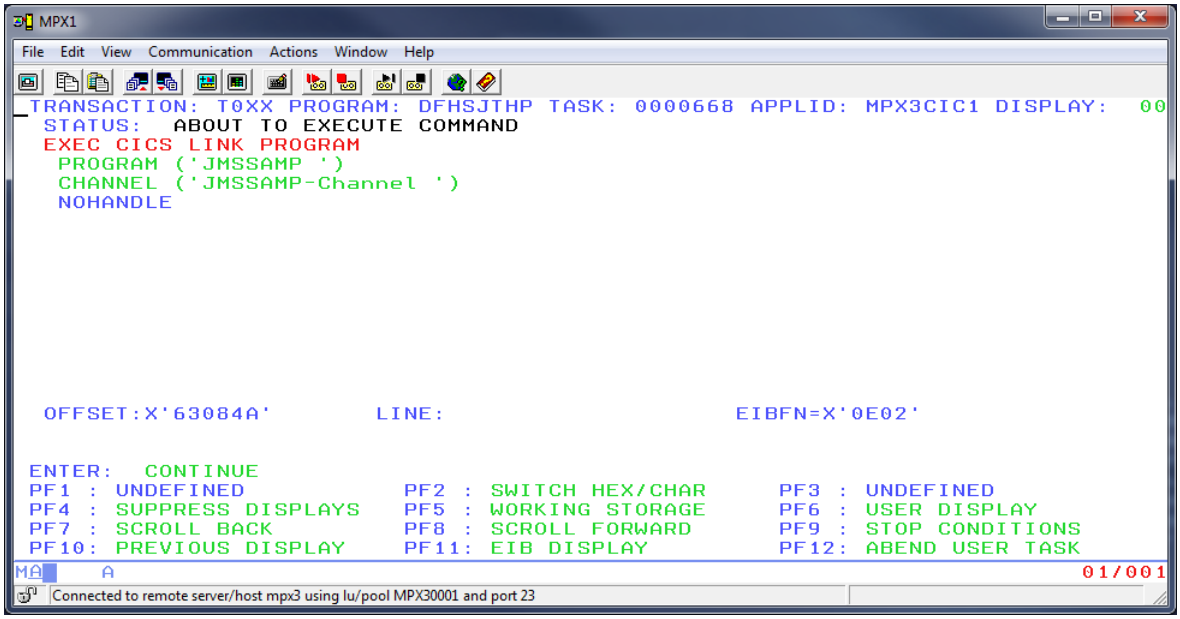


___ 13. Press the **Execute** button in the browser. This action will cause transaction *T0XX* to be started and an EDF session started in the CICS 3270 session.

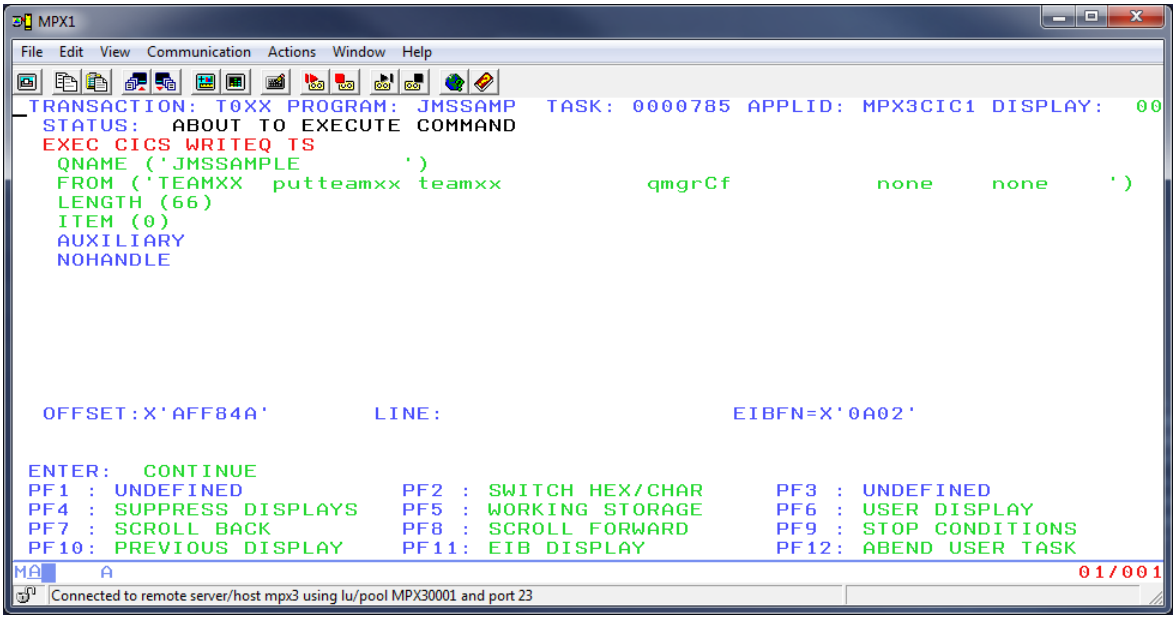


Tech-Tip: Once EDF is active in CICS, the web browser application will wait for user interactions before continuing. If you see the web browser waiting for a response check to see if there is a CICS EDF screen waiting for an Enter key response. You may have to keep pressing Enter before the web browser will render an update or terminate the EDF session by pressing F3.

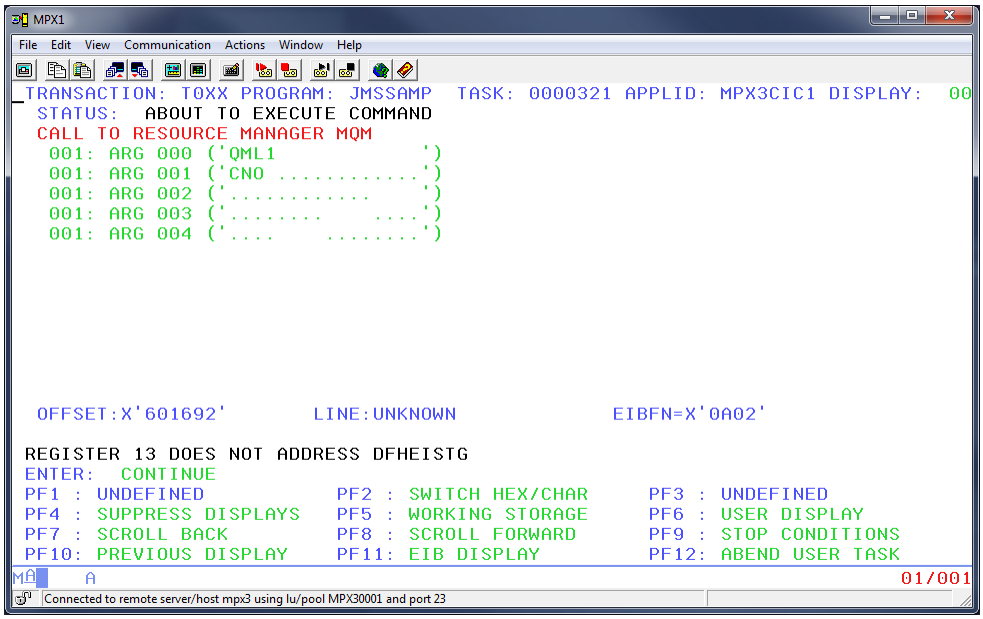
___ 14. Step though the EDF screens in the CICS session by pressing the **Enter** key until you see the screen below. This screen appears as a result of the JCICS *task.link* performed in *JMSResults.java*



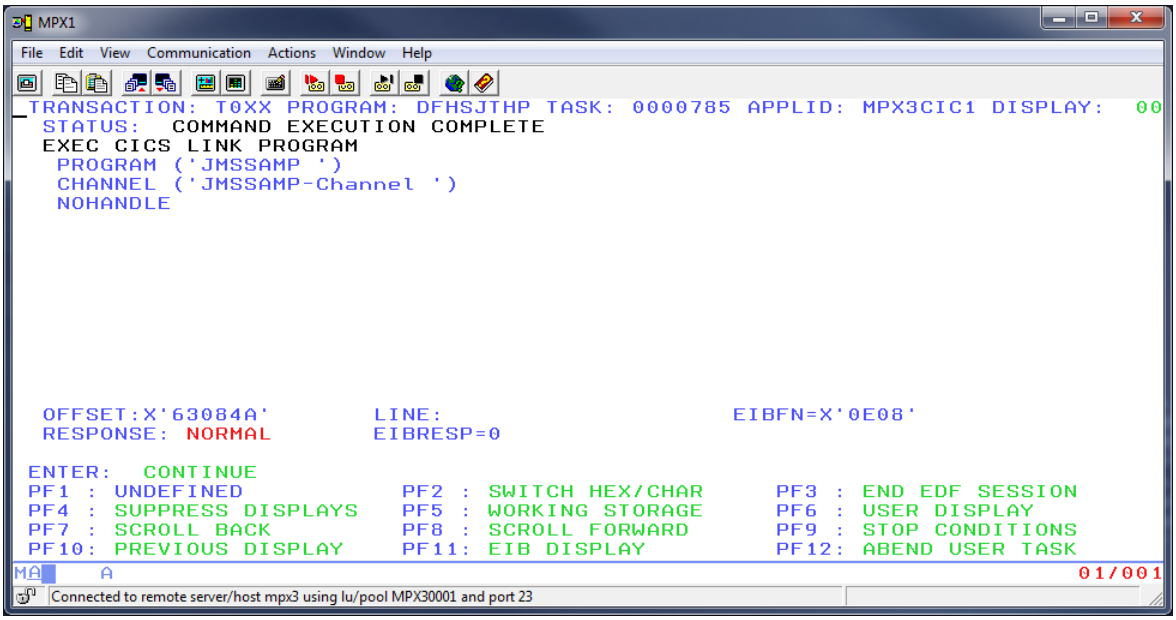
___ 15. Press **Enter** again until the screen below appears. This is the caused by the JCICS *tsqQ.writeString* request in *JMSSample.java*.



___ 16. Press **Enter** again until the screen below appears. This shows that the JMS connection request is about to be made to the queue manager. Note that JMS connection request to queue managers are basically ignored since CICS handles the connection to the queue manager on behalf of the application.

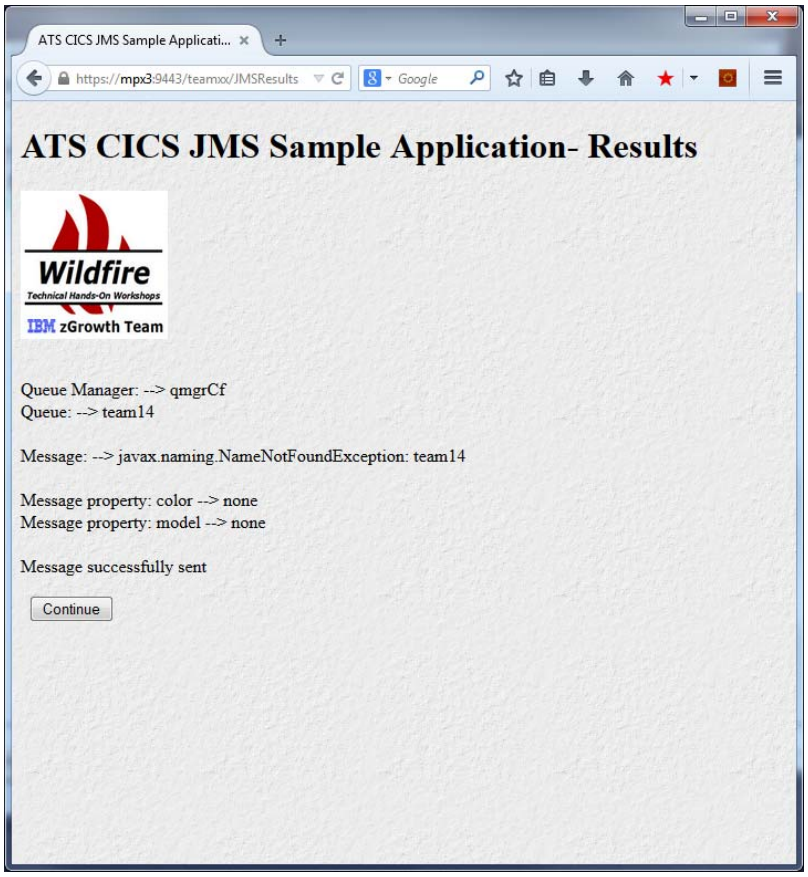


___ 17. Continue pressing **Enter** and you will see subsequent JMS requests and CICS syncpoint requests all driven by the execution of program JMSSAMP. Eventually you will see a screen below where the JMS enable CICS program is returning to the web application.



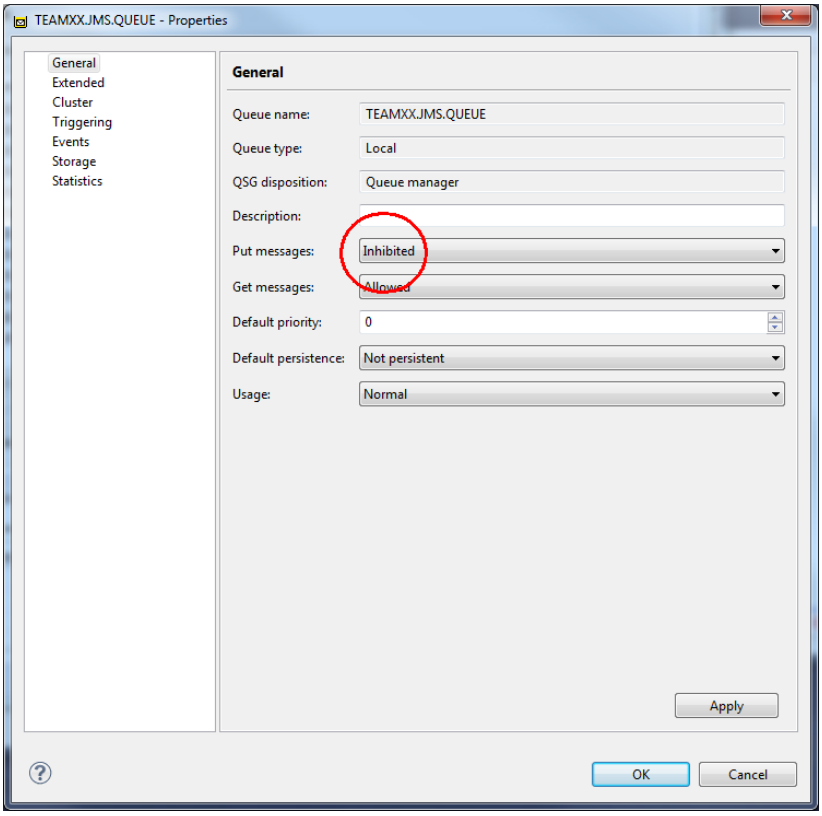
___ 18. Press the **F3** key to terminate EDF tracing. While EDF is active the web browser is waiting and using EDF should be used only when necessary.

___ 19. Once you are satisfied that the application is executing as expected select another team's queue on the *JMS Sample Application* screen and try to write a message to that queue. What happened?

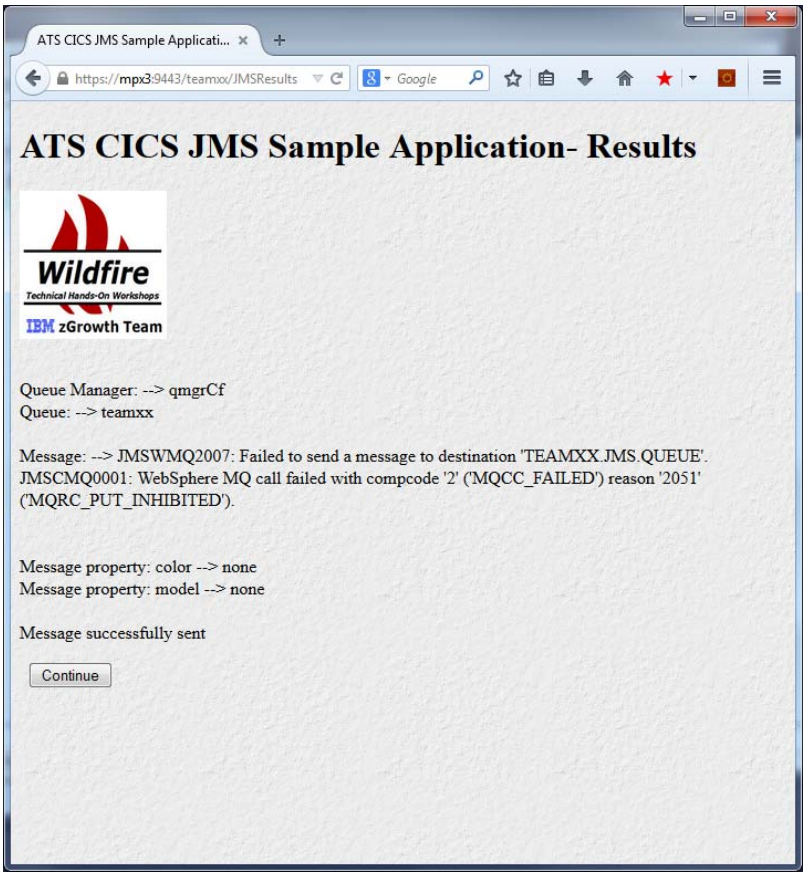


___ 20. The Naming exception occurs because the JNDI bindings file does not contain the other team's destination. So when the application used the other team's queue name there was no information for that queue for JNDI service provider to provide.

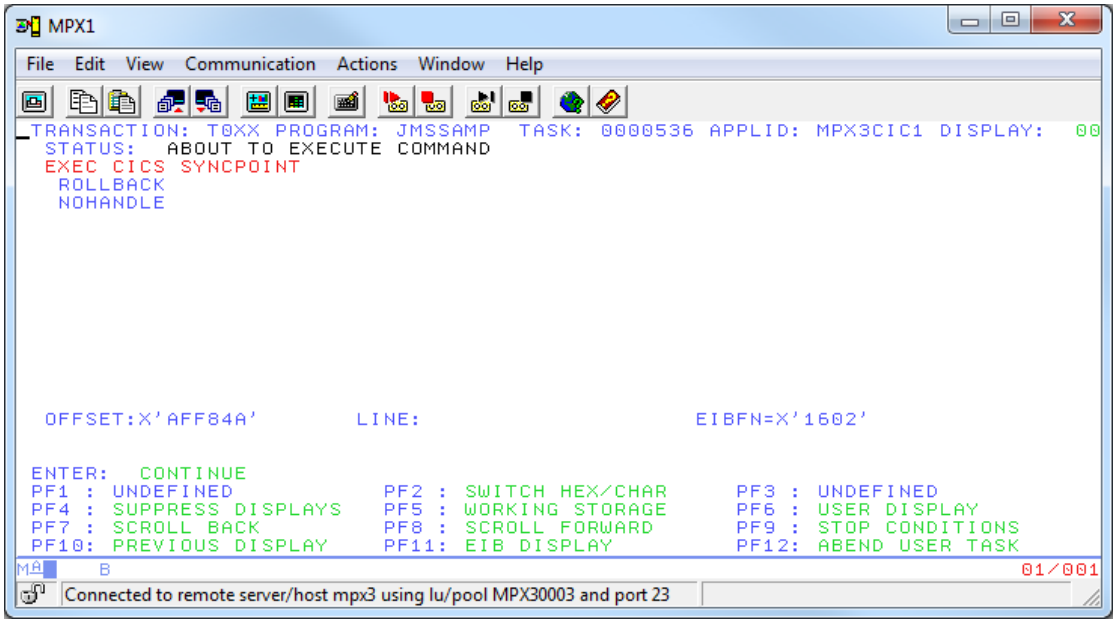
___ 21. Next use MQ Explorer to change your queue so that puts and/or gets are inhibited and retry JMSSample.



___ 22. The MQ failure should be reason code indicate that the queue was either put or get inhibited. See an example of a put inhibited failure below.



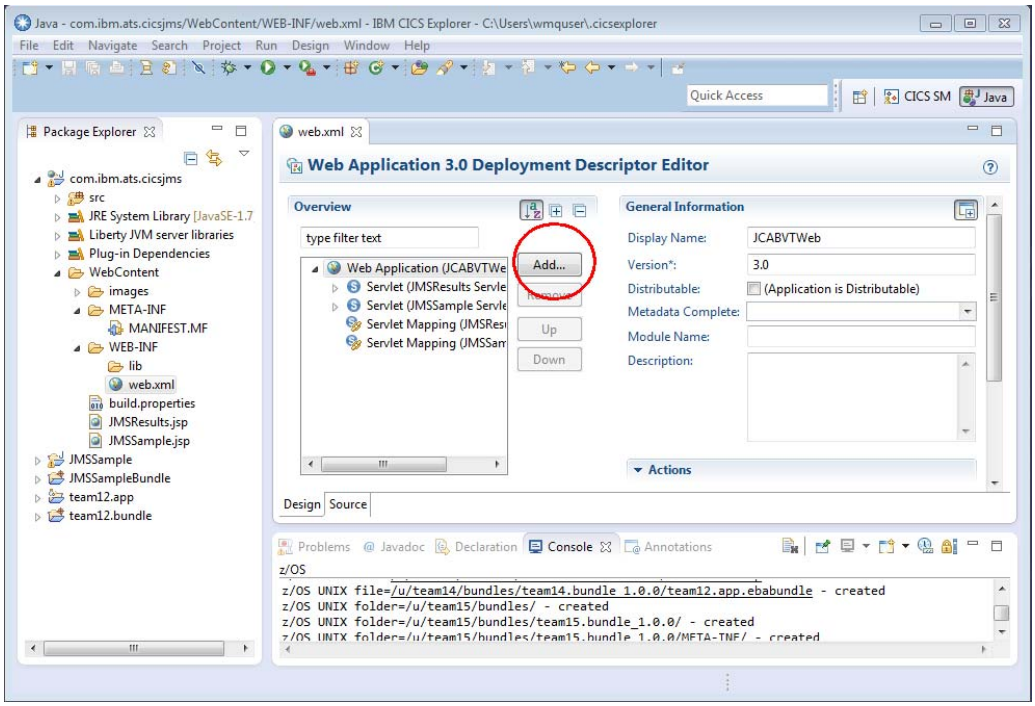
___ 23. Start an EDF trace of transaction T0XX again and this time you should see the screen below. This corresponds to the invoking of the JCICS *task.rollback* method .



Part 5 – Enabling Liberty Security (Optional)

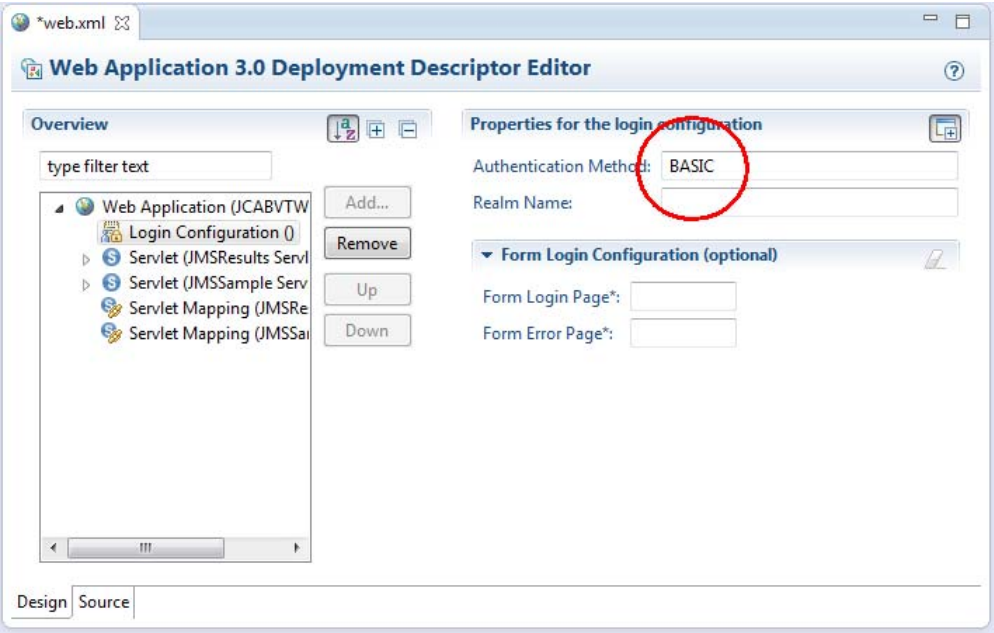
Enabling security for the web application requires adding additional information to the application's *web.xml* file as well as configuration changes to CICS Liberty artifacts. The former of these changes will be done in this part of the exercise. The changes to CICS Liberty artifacts have already been done and are documented in the Appendix.

___ 1. Expand the *com.ibm.ats.cics.jms* package and then expand *WebContent* and then *WEB-INF* to expose the *web-xml* file. Double click this file or open it with the *Web Application 3.0 Deployment Descriptor Editor* (see below).



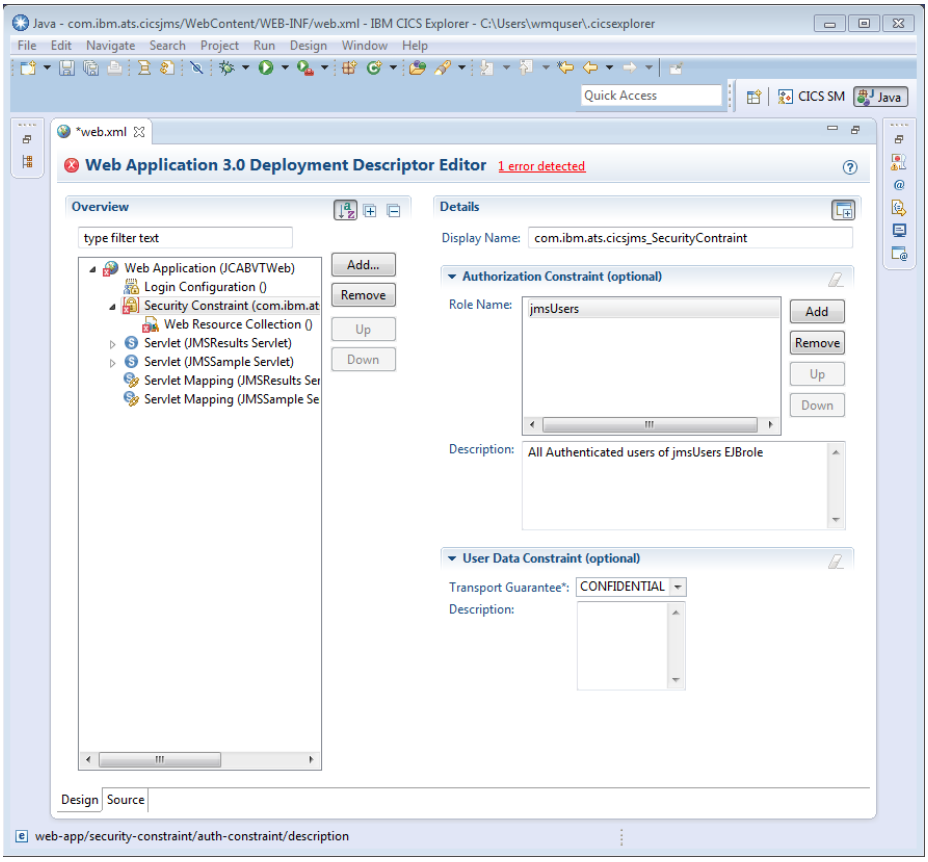
___ 2. Click the **Add** button and selection *Login Configuration* from the list on the *Add Item* screen and press **OK** to continue.

___ 3. In the *Properties for the login configuration* section of the screen enter ***BASIC*** in the area beside *Authentication Method*.

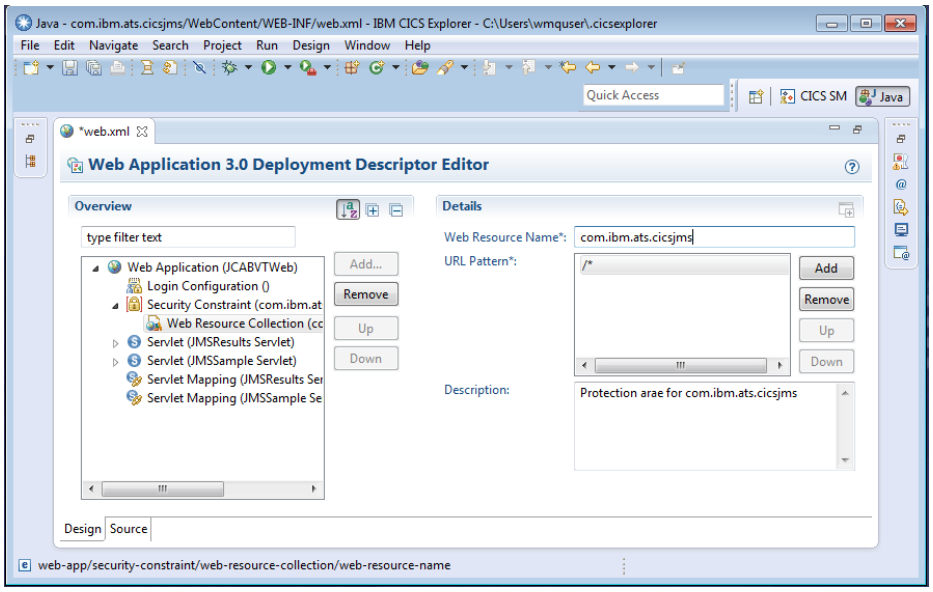


___ 4. Select *Web Application* in the *Overview* section of the screen and press the **Add** button again. This time add a *Security Constraint* to the application.

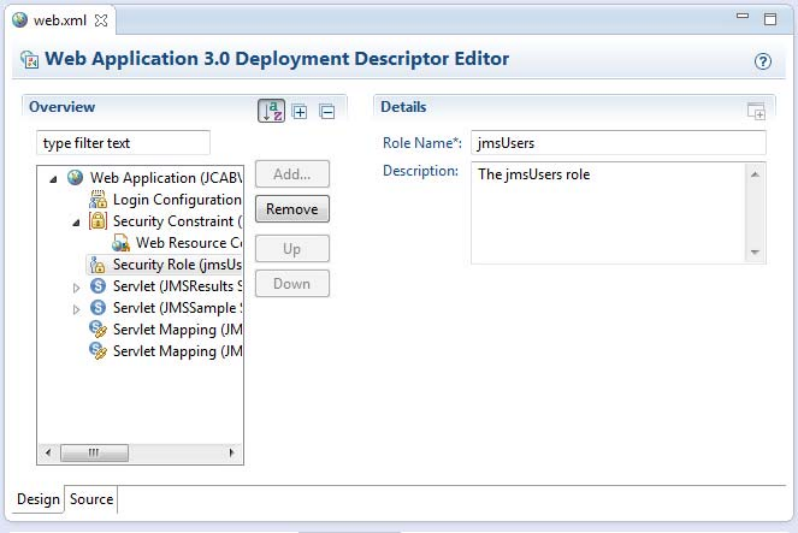
___ 5. Enter ***com.ibm.ats.cicsjms_SecurityConstraint*** as the *Display Name* and ***All Authenticated users of jmsUsers EJBRole*** as the *Description*. Click the **Add** button under *Authorization Constraints* to add a *Role Name* of ***jmsUsers***. Use the pull down arrow to select a transport guarantee type of *CONFIDENTIAL*.



___ 6. Select *Web Resource Collection* under *Security Constraints* and enter ***com.ibm.ats.cicsjms*** as the *Web Resource Name* and ***Protection area for com.ibm.ats.cicsjms*** as the *Description*. Press the **Add** button as enter **/*** as the *URL Pattern*.



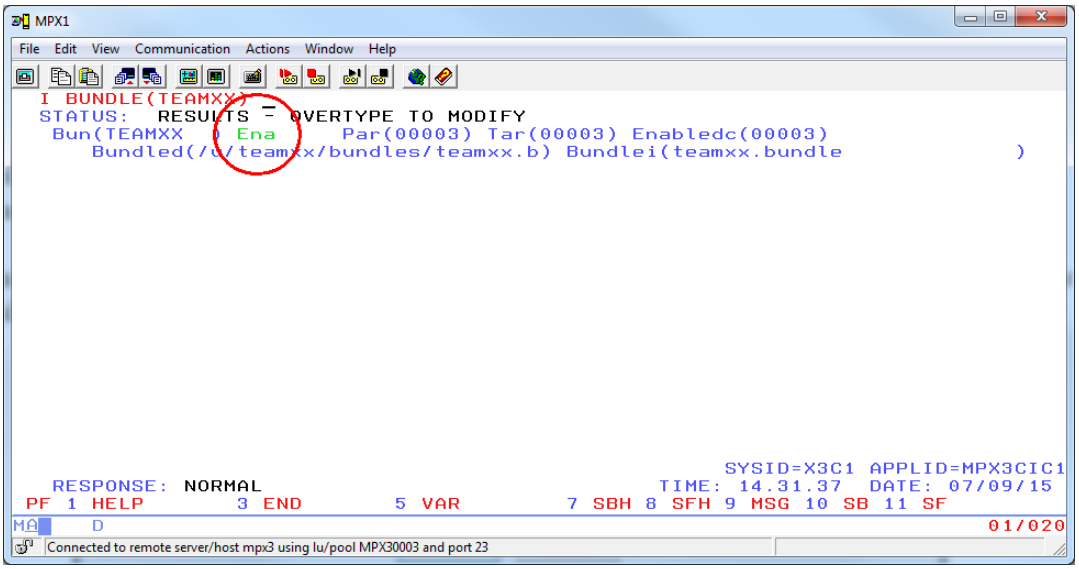
___ 7. Select *Web Application* in the *Overview* section of the screen and press the **Add** button again. This time add a *Security Role* to the application. Enter ***jmsUsers*** as the *Role Name* and ***The jmsUsers*** role as the *Description* (see below).



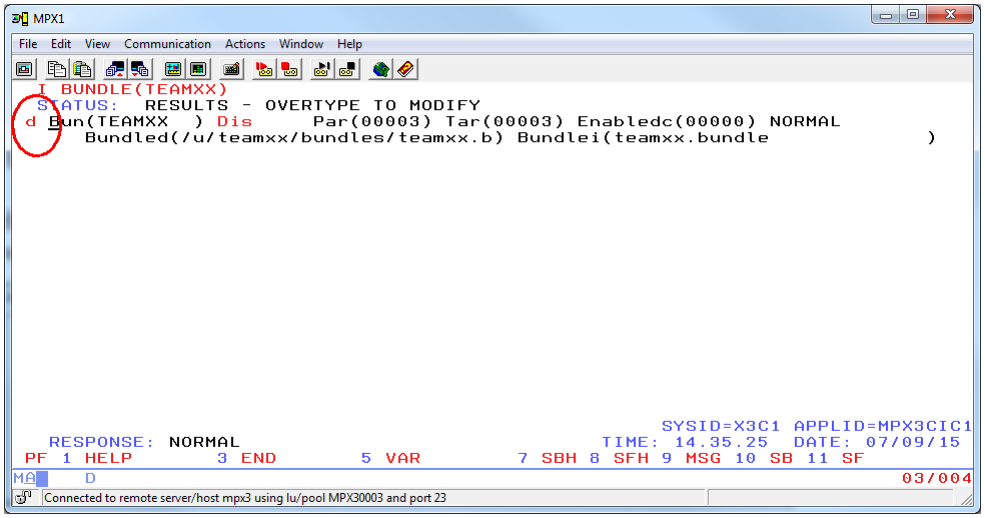
___ ***8.*** Save all changes and export the updated bundle (*teamxx.bundle*) to z/OS (see Steps 37 through 39 in Part 3).

___ ***9.*** In a CICS 3270 session enter CICS transaction ***CEMT I BUNDLE(TEAMXX)*** to display the bundle containing the JMS web application.

___ ***10.*** Before a bundle can be re-installed in CICS it must be disabled and discarded. Overtype the *Ena* area of the screen with ***DIS*** to disable the bundle. Press **Enter** to continue.

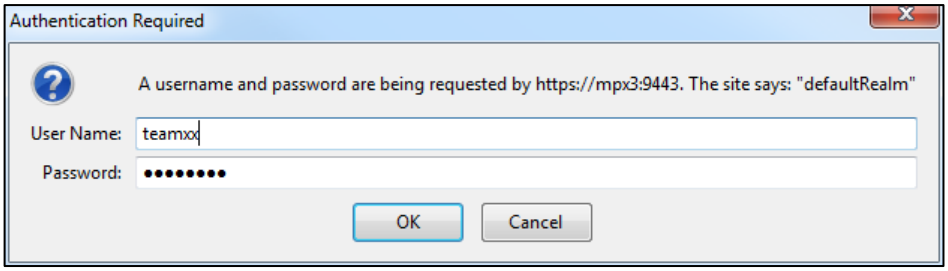


___ 11. To discard the bundle, enter a **D** in the area before the bundle and press **Enter**.

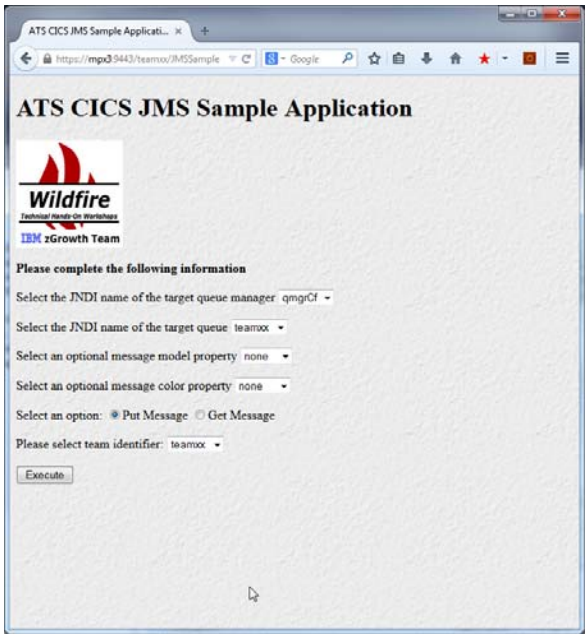


___ 12. Terminate CEMT by pressing the **F3** key. Clear the screen and enter CICS transaction ***CEDA IN G(JMSSAMP) BUNDLE(TEAMXX)*** to install the bundle containing the JMS web application.

___ 13. Close any open browser sessions and access the JMS web application by opening a new web browser session and entering URL ***mpx1:9443/teamxx/JMSSample***. This time an *Authentication Required* window should appear prompting for a *User Name* and *Password*.



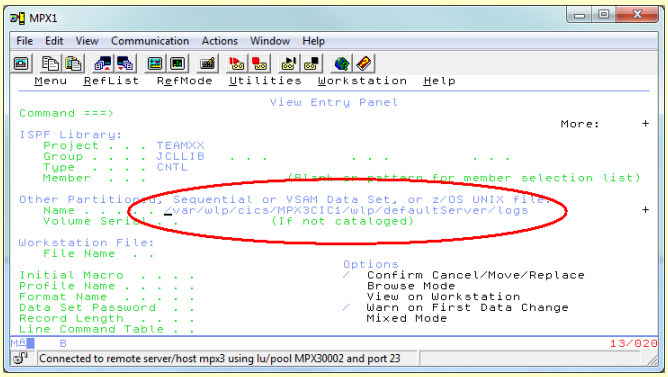
___ 14. Enter the *userid* and *password* from the worksheet and press **OK**. The *ATS JMS Sample Application* should appear as before.



___ 15. Close the browser entirely and retry but this time logon as USER1 using USER1 as the password. The logon prompt reappears because logon attempt fails even though the password is valid. The failure is because user does not have access to the RACF resource BBGZDFLT in the APPL class. The messages below will appear in the CICS JVM Liberty server's messages log file.

SAF Service IRRSIA00_CREATE did not succeed because user USER1 has insufficient authority to access APPL-ID BBGZDFLT. SAF return code 0x00000008. RACF reason code 0x00000020. Authentication did not succeed for user ID TEAM11. An invalid user ID or password was specified.

Tech-Tip: CICS JVM Liberty messages can be viewed by using ISPF option 1 (browse) or 2 (edit) and specifying directory ***/var/wlp/cics/MPX3CIC1/wlp/defaultServer/logs*** in the area beside *Name* under *Other Partitioned, Sequential or VSAM Data Set, or z/OS Unix file* (see below). This will display a file with a name of *message.log.* This file is written in ASCII so to view its contents enter command ***VA*** beside the file name and press **Enter**.

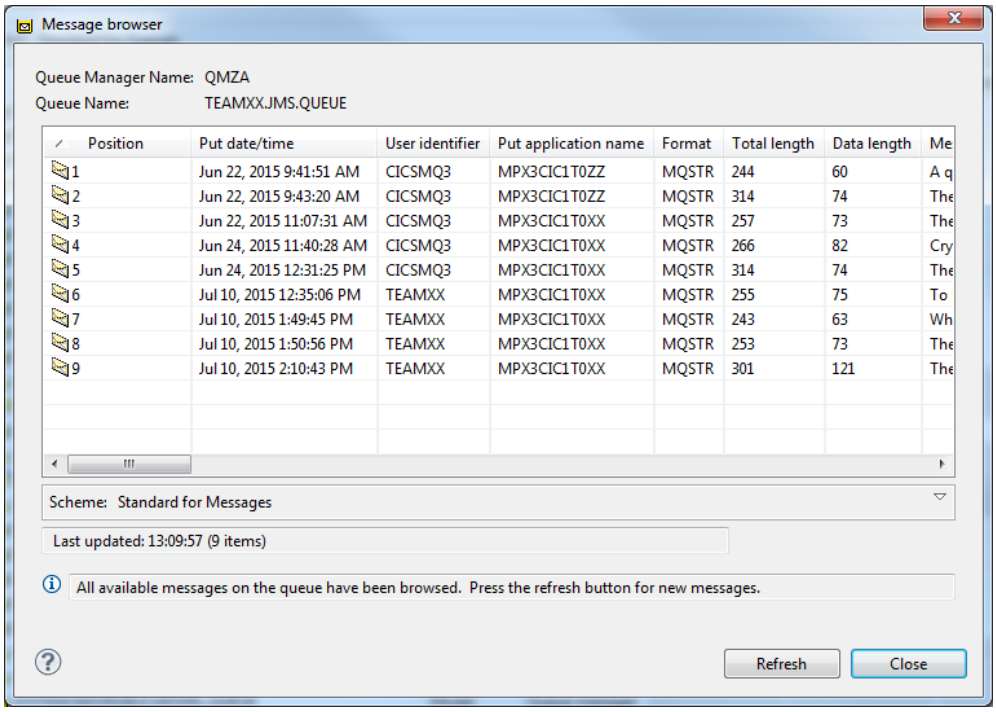


___ 16. Try logging on as USER2 using USER2 as the password. You see message *Error 403: AuthorizationFailed*. This message occurs because user identity USER2 does not have access to RACF EJBRole resource *BBGZDFLT.teamxx.app.jmsUsers*. The messages below will appear in the CICS JVM Liberty server's messages log file.

```
CWWKS9104A: Authorization failed for user user2 while invoking teamxx.app on /JMSSample.
The user is not granted access to any of the required roles: [jmsUsers].
```

Tech-Tip: The RACF EJBRole name is determined by value of the safRoleMapper parameter in the Liberty *server.xml* file and the name of the *Enterprise Bundle Archive bundle* (EBABUNDLE) defined in the CICS bundle (see *cics.xml* in *teamxx.bundle)*

___ 17. Finally, use the MQExplorer to browse the messages in the destination queue. Notice that the messages written when security was not enabled have the CICS default userid as the message's *User identifier* while the messages written with security enabled have the userid entered on the authentication screen as the message's *User identifier*.



This completes this exercise.

Appendix – CICS Liberty Configuration Files

This section contains the various configuration files used by CICS OSGi and CICS Liberty JVM servers. Also included is a section for the RACF command performed to configure security.

CSCJVM.jvmprofile

CSCJVM.jvmprofile provides the configuration information for the CICS JVMServer CSCJVM. This is only an excerpt of the contents of CSCJVM.jvmprofile.

```
# JAVA_HOME specifies the location of the Java directory.
#
JAVA_HOME=/shared/java/J7.0_64/

# Specify the location for CICS JVM Server to write
# the STDIN, STDOUT and STDERR streams to
WORK_DIR=/var/wlp/cics/MPX1CIC1 [1]

#
LIBPATH_SUFFIX=/shared/db2a10/jdbc/lib:/usr/lpp/mqm/V8R0M0/java/lib
#
OSGI_BUNDLES=/usr/lpp/mqm/V8R0M0/java/lib/OSGi/com.ibm.mq.osgi.allclientprereqs_8.0.0.2.jar,\ [2]
        /usr/lpp/mqm/V8R0M0/java/lib/OSGi/com.ibm.mq.osgi.allclient_8.0.0.2.jar,\
        /var/wlp/cics/lib/com.ibm.etools_1.0.0.jar,\
        /var/wlp/cics/lib/javax.resource_1.0.0.jar,\
        /shared/cicsts/cicsts52/lib/dfjrouter.jar
OSGI_FRAMEWORK_TIMEOUT=60
STDOUT=./cics/output/dfhjvmout.&JVMSERVER;.&APPLID;.data [3]
STDERR=./cics/output/dfhjvmerr.&JVMSERVER;.&APPLID;.data [4]
JVMTRACE=/dev/null
PRINT_JVM_OPTIONS=YES
```

1. Identifies the OMVS directory where this JVMServer will use for configuration files, logs, error messages, etc.
2. Identifies the OSGi Jar file bundles required for the CICS Java application.
3. Identifies the standard Java output (STDOUT) file (within WORK_DIR)
4. Identifies the standard Java error message (STDERR) file (within WORK_DIR)

Tech-Tip: OMVS files *CSCJVM.profile* and *CSCWLP.profile* are in the directory specified by CICS SIT parameter JVMPROFILEDIR.

CSCWLP.jvmprofile

CSCWLP.jvmprofile provides the initial configuration information for the CICS JVMServer CSCWLP. This is only an excerpt of the contents of CSCWLP.jvmprofile.

```
JAVA_HOME=/shared/java/J7.0_64/
WORK_DIR=/var/wlp/cics/MPX1CIC1 [1]
WLP_INSTALL_DIR=/shared/cicsts/cicsts52/wlp [2]
-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location=/shared/db2a10/jdbc
PRINT_JVM_OPTIONS=YES
WLP_OUTPUT_DIR=./wlp/ [3]
-Dcom.ibm.cics.jvmserver.wlp.server.name=defaultServer [4]
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=false [5]
-Dcom.ibm.cics.jvmserver.wlp.server.host=*
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=1591
-Dcom.ibm.cics.jvmserver.wlp.server.https.port=9443
STDOUT=./cics/output/dfhjvmout.&JVMSERVER;.&APPLID;.data [6]
STDERR=./cics/output/dfhjvmerr.&JVMSERVER;.&APPLID;.data [7]
JVMTRACE=/dev/null
```

1. Identifies the OMVS directory where this JVMServer will use for configuration files, logs, error messages, etc.
2. Identifies the OMVS directory where the CICS provided Liberty executables can be found.
3. Identifies the OMVS directory (within WORK_DIR) that the CICS JVM Liberty server will use for its purposes.
4. The CICS JVM Liberty server name
5. Disables the automatic configuration of the server.xml file
6. Identifies the standard Java output (STDOUT) file (within WORK_DIR)
7. Identifies the standard Java error message (STDERR) file (within WORK_DIR)

Tech-Tip: Using the above as an example:

CICS JVM Liberty messages would be written in ASCII to file
/var/wlp/CICS/MPX1CIC1/wlp/defaultServer/logs/messages.log

Java standard messages would be written in EBCDIC to file
/var/wlp/cics/MPX1CIC1/cics/output/dfhjvmout.CSCWLP.MPX3CIC1.data

Java standard error messages would be written in EBCDIC to file
/var/wlp/cics/MPX1CIC1/cics/output /dfhjvmerr.CSCWLP.MPX3CIC1.data

The OMVS file *server.xml* used by the CICS JVM Liberty server would be located in directory
/var/wlp/cics/MPX1CIC1/MPX3CICS1/CSCWLP/wlp/usr/servers/defaultServer

server.xml

Below is an excerpt from the *server.xml* that shows the configuration details relevant for this exercise.

```
    <!-- Enable features -->
    <featureManager>
        <feature>cicsts:core-1.0</feature>
        <feature>jsp-2.2</feature>
        <feature>wab-1.0</feature>
        <feature>blueprint-1.0</feature>
        <feature>cicsts:security-1.0</feature>
        <feature>cicsts:jdbc-1.0</feature>
        <feature>ssl-1.0</feature>
    </featureManager>

    <!-- Default HTTP End Point -->
    <httpEndpoint host="*" httpPort="1591" httpsPort="9443" id="defaultHttpEndpoint"/>

  <bundleRepository>
   <fileset dir="/var/wlp/cics/lib" include="*.jar"/>
  </bundleRepository>

  <safRegistry id="saf" />
  <safAuthorization id="saf"/>
  <safCredentials profilePrefix="BBGZDFLT"/>
  <safCredentials unauthenticatedUser="WSGUEST"/>
  <safRoleMapper profilePattern="BBGZDFLT.%resource%.%role%"/>
```

Tech-Tip: The value for the SAF variables *profilePrefix* and *profilePattern* (BBGZDFLT) defaults to BBGZDFLT. This value must match the values used in the commands when defining some resources to RACF in order for the proper security checks to be performed, see RACF commands in the appendix.

RACF Commands

Below are samples commands used to define and grant permission to the RACF resources required for CICS JVM Liberty server. Note that the default security prefix, e.g. BBGZDFLT (red to provide emphasis) is configurable and much match the prefix used in the CICS Liberty configuration file *server.xml*.

In the sample commands below:

CICSUSER is the RACF identity of the CICS region
CICSLIBR is a group of user identifies that have access to CICS Liberty
OTHRUSRS is a group of users that have access to CICS but not CICS Liberty

```
/* Grant access to the default security prefix BBGZDFLT
/* Users not granted access to this resource will not be able to be authenticated by CICS or Liberty.
RDEFINE APPL BBGZDFLT OWNER(SYS1)
PERMIT BBGZDFLT CLASS(APPL) RESET
PERMIT BBGZDFLT CLASS(APPL) ID(CICSLIBR,CICSUSER,WSGUEST) ACCESS(READ)
PERMIT BBGZDFLT CLASS(APPL) ID(OTHRUSRS) ACCESS(READ)

/* Grant access to the Liberty angel process to the CICS Liberty process (required to access z/OS Authorized services)
RDEFINE SERVER BBG.ANGEL OWNER(SYS1)
PERMIT BBG.ANGEL CLASS(SERVER) RESET
PERMIT BBG.ANGEL CLASS(SERVER) ID(CICSUSER) ACC(READ)

/* Grant access to use  z/OS Authorized services  to the CICS Liberty process
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) RESET
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ID(CICSUSER) ACC(READ)

/* Grant access to RACF authorization services to the CICS Liberty process
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) RESET
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ID(CICSUSER) ACC(READ)

/* Grant permission to perform authentication for the default security prefix BBGZDFLT to the CICS Liberty process
RDEFINE SERVER BBG.SECPFX.BBGZDFLT OWNER(SYS1)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) RESET
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ID(CICSUSER) ACC(READ)

/* Grant access to IFAUSGE (SMF)  to the CICS Liberty process
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.PROGMGR CLASS(SERVER) RESET
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ID(CICSUSER) ACC(READ)

/* Grant access to the application using application's EJBRole
RDEFINE EJBROLE BBGZDFLT.teamxx.app.jmsUsers OWNER(SYS1)
PERMIT BBGZDFLT.teamxx.app.jmsUsers CLASS(EJBROLE) RESET
PERMIT BBGZDFLT.teamxx.app.jmsUsers CLASS(EJBROLE) ID(TEAMXX) ACC(READ)

SETROPTS RACLIST(SERVER,APPL,EJBROLE) REFRESH
```

CICS Bundle Definition

Below is a display showing the bundle resource definition defined in CICS.

```
MPX1
File  Edit  View  Communication  Actions  Window  Help

 OBJECT CHARACTERISTICS                                    CICS RELEASE = 0690
  CEDC  View Bundle( TEAMXX    )
   Bundle          : TEAMXX
   Group           : JMSSAMP
   DEScription     :
   Status          : Enabled            Enabled | Disabled
   BUndledir       : /u/teamxx/bundles/teamxx.bundle_1.0.0
   (Mixed Case)    :
                   :
                   :
                   :
   BAsescope       :
   (Mixed Case)    :
                   :
                   :
                   :
  DEFINITION SIGNATURE
   DEFinetime      : 06/19/15 09:37:32
+  CHANGETime      : 06/19/15 14:54:38

                                                   SYSID=X2C1 APPLID=MPX2CIC1

 PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
MA    D                                                                 01/003
Connected to remote server/host mpx2 using lu/pool MPX20002 and port 23
```

And the same definition displayed in CICS Explorer.

Bundle Definition (TEAMXX)

Attributes

MPX2CIC1 ▸ MPX2CIC1 ▸ TEAMXX

type here to filter on Name and CICS Name

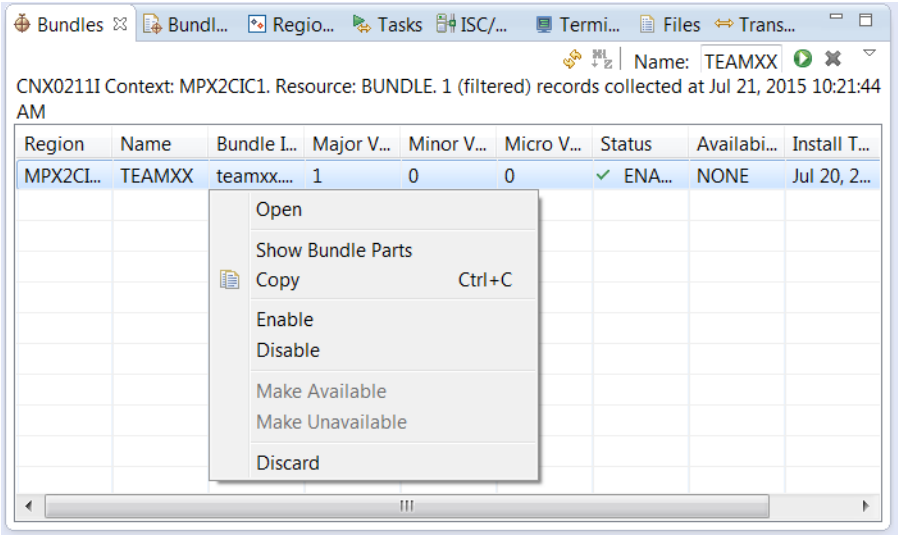
Name	CICS Name	Value
Basic		
Basescope	BASESCOPE	
Bundle Directory	BUNDLEDIR	/u/teamxx/bundles/teamxx.bundle_1.0.0
CSDGroup	CSDGROUP	JMSSAMP
Description	DESCRIPTION	
Name	NAME	TEAMXX
Status	STATUS	ENABLED
Version	DEFVER	0
Definition Signature		
Change Agent	CHANGEAGENT	CSDAPI
Change Release	CHANGEAGREL	0690
Change Time	CHANGETIME	Jun 19, 2015 2:54:38 PM
Change User ID	CHANGEUSRID	CICSMQ3
Create Time	CREATETIME	Jun 19, 2015 9:37:32 AM

Attributes

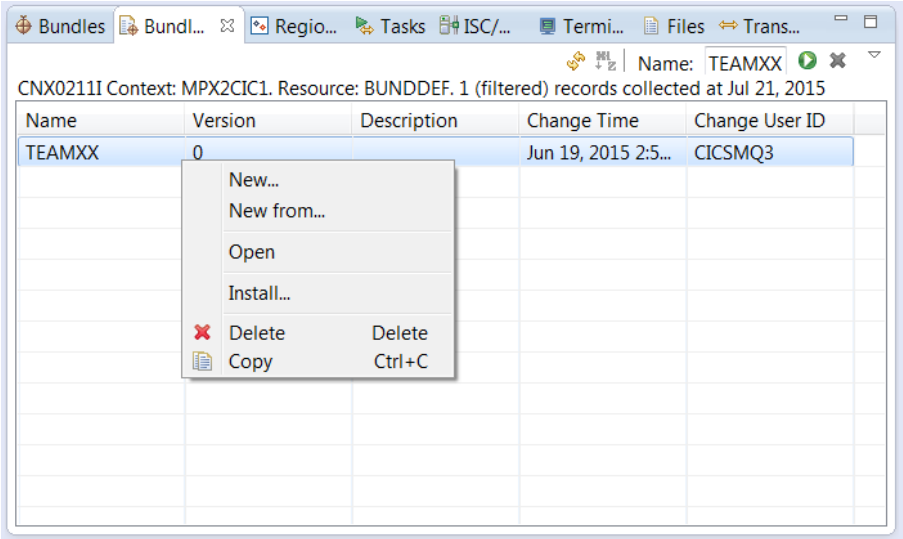
CICS Bundle Management

CICS bundles can be managed from CICS Explorer in the CICS SM perspective by selecting the *Operations* tool in the tool bar and selecting *Bundles* from the list of CICS resources. In the examples below the filtering option was used to display only the TEAMXX bundle.

Once the bundle is displayed it can be selected and right mouse button can be used to *Disable* the bundle and then *Discard* it.



Once the bundle has been discarded an updated bundle can be installed by selecting the *Definitions* tool in the tool bar and selecting *Bundle Definitions* from the list of CICS resources



Summary

- The exercise began by reviewing some basic JMS terms and concepts.
- The next step used MQ Explorer to configure the JNDI configuration to create a .bindings file which was moved to a z/OS OMVS directory which CICS could access.
- CICS Explorer was then used to make the web application unique and to review the CICS JMS enable application. CICS Explorer was then used to deploy and install the bundle containing the web application.
- The application was then tested and verified using MQ Explorer and CICS execution diagnostic facility.
- CICS Liberty security was enabled and combinations of valid and invalid authentication attempts were made and the results reviewed.