Introduction to IBM MQ on z/OS JMS Support for CICS and Liberty

Lyn Elkins – **elkinsc@us.ibm.com**
Mitch Johnson – **mitchj@us.ibm.com**

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

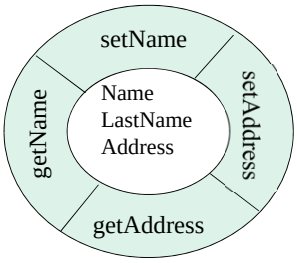
JMS – Java Message Service

- JMS is the industry standard Java API for messaging
  - point-to-point messaging domain
  - publish/subscribe messaging domain
- Vendor-independent Messaging API in Java
  - Specification owned by Oracle
  - Managed by The Java Community Process
  - Expert Group includes IBM, RedHat, et. al.
- Part of Java Enterprise Edition standard
  - Uses Java Naming and Directory Interface (JNDI)
- Defines the package of common Java Interfaces
  - Provides provider-independence
  - Does not provide provider interoperability between providers

Basic Java Messing Service Programming

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

Quick Comparison of a Java Object v. COBOL




```
01 Customer
    10 Name        PIC X(20)
    10 LastName    PIC X(20)
    10 Address     PIC X(40).
```

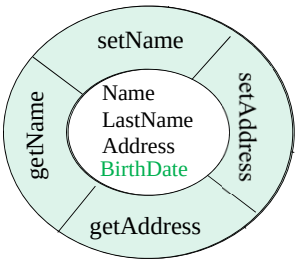
```
Customer customer = new Customer();
customer.setName("John");
newAddress = customer.getAddress();
```

Name of Customer = 'John'.
Address = Address of Customer.

- *setAddress* and *getAddress*, *etc.* are methods that either retrieves or changes the contents of an instance variable.

Key Object Oriented Point - Encapsulation: Java encapsulates the data inside an 'object' and hides the implementation details from the users of that object. Therefore if the implementation for accessing the data needs to change, the user is not impacted.

Quick Comparison of a Java Object v. COBOL



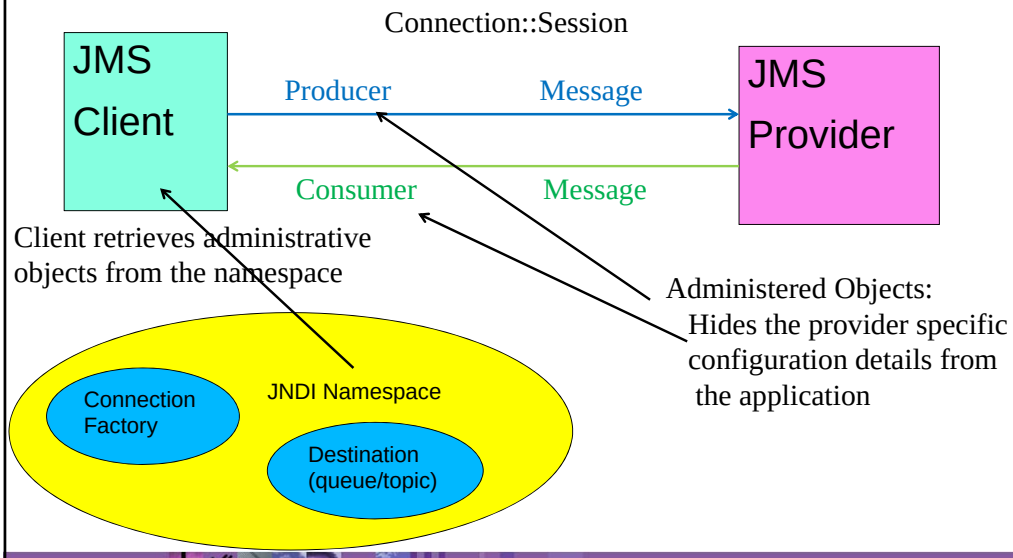

```
Customer customer = new Customer();
customer.setName("John");
String Address = customer.getAddress();
String fullName = customer.getFullName();
String birthDate = setBirthDate("01/10/1980");
```

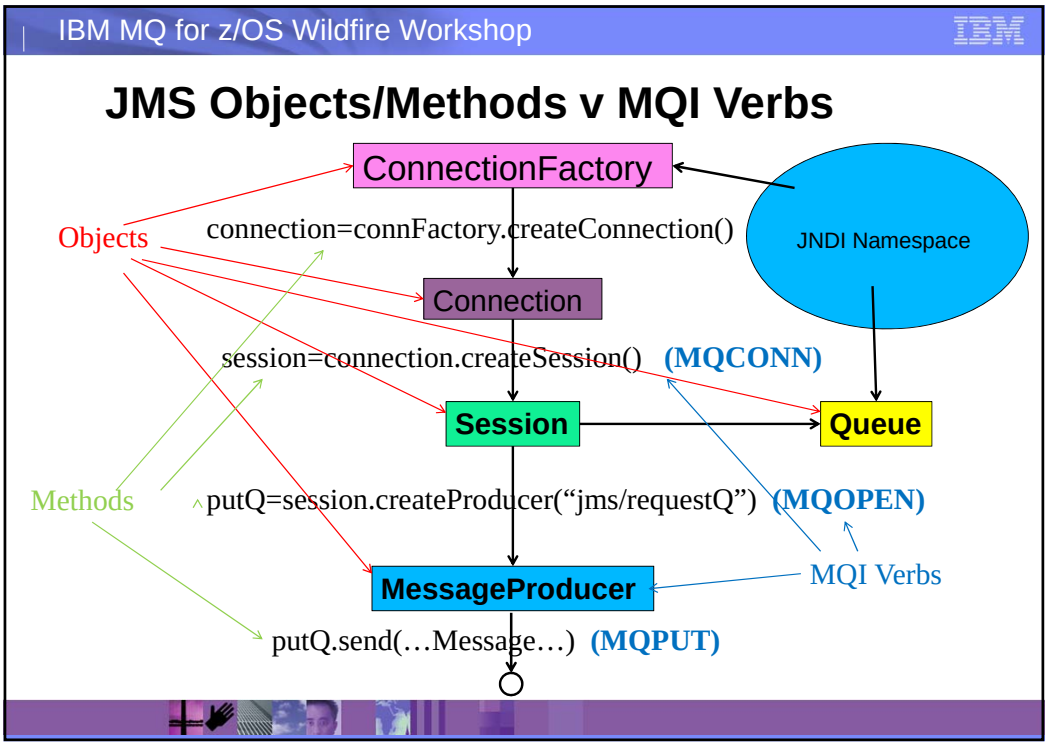
```
01 Customer
    10 Name        PIC X(20)
    10 LastName    PIC X(20)
    10 Address     PIC X(40).
    10 BirthDate   PIC X(10).
 01  FullName      PIC X(40).
```

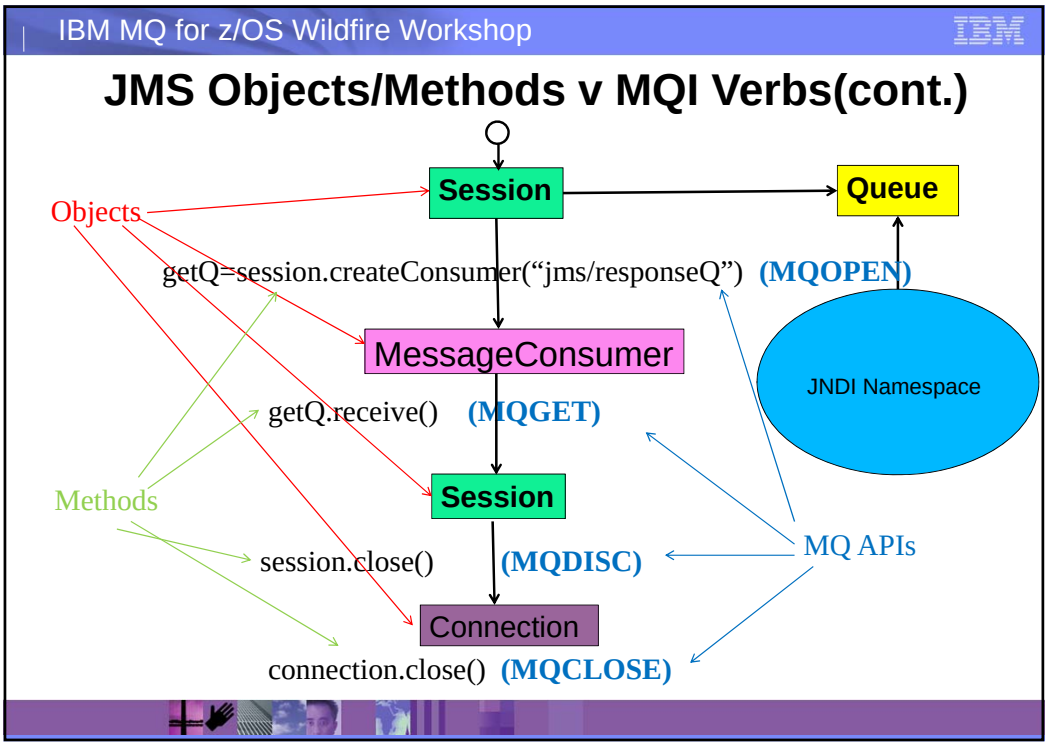
Name of Customer = 'John'.
Address = Address of Customer.
String Name of Customer Delimited by Space
LastName of Customer Delimited by Space
into FullName.
BirthDate of Customer = "01/01/1980".

- *###BirthDate* and represents the getter and setter methods that either set or change the value of instance variable *BirthDate*.
- *getFullName* is a method that concatenates *Name* and *LastName* and returns the full name of the customer.

JMS uses Java Objects for messaging




IBM MQ for z/OS Wildfire Workshop
IBM
JMS Objects/Methods v MQI Verbs
ConnectionFactory
Objects
connection=connFactory.createConnection()
JNDI Namespace
Connection
session=connection.createSession() (MQCONN)
Session
Queue
Methods
putQ=session.createProducer("jms/requestQ") (MQOPEN)
MQI Verbs
MessageProducer
putQ.send(…Message…) (MQPUT)


IBM MQ for z/OS Wildfire Workshop
IBM
JMS Objects/Methods v MQI Verbs(cont.)
Session
Queue
Objects
getQ=session.createConsumer("jms/responseQ") (MQOPEN)
MessageConsumer
JNDI Namespace
getQ.receive() (MQGET)
Methods
Session
MQ APIs
session.close() (MQDISC)
Connection
connection.close() (MQCLOSE)

JMS Sample Code

```
// Instantiate the initial context
Context initContext = new InitialContext();
// Lookup and retrieve a Connection Factory from the name space
ConnectionFactory connFactory = (ConnectionFactory)  initContext.lookup("jms/qmgr");
// Create a Connection object using the factory (based on information obtained from the name space)
   javax.jms.Connection thisConnection = connFactory.createConnection();
// Start the connection to the queue manager using the connection object
thisConnection.start();

// Create a Session object using the connection object
Session thisSession =  conn.createSession(false, Session.AUTO_ACKNOWLEDGE);
// Lookup and retrieve the Destination (queue) information from the name space
Destination putQueue = (Destination) context.lookup("jms/requestQueue");
Destination getQueue = (Destination) context.lookup("jms/responseQueue");
// Create producer/consumer objects using the session and destination objects
MessageProducer msgProducer = (MessageProducer) thisSession.createProducer(putQueue);
MessageConsumer msgConsumer = (MessageConsumer) thisSession.createConsumer(getQueue);
// Send and receive message using the producer/consumer  objects
TextMessage message = (TextMessage) thisSession.createTextMessage();
msgProducer.send(message);
message = (TextMessage) msgConsumer.receive();
```

JMS Objects/Methods and COBOL

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

First Obtain a Connection Factory

- A JMS connection factory is obtained by doing an indirect JNDI lookup of the queue manager's connection factory
  - *ConnectionFactory connectionFactory = (ConnectionFactory) context.lookup('jms/qmgr');*
- The connectionFactory instance object is populated with information from the name space, such as:
  - *Queue Manager name*
  - *Transport type: bindings or client*
  - *Port*
  - *Host name*
  - *Client Channel*
  - *SSL information*

Create a connection to a Queue Manager

- Use the returned connection factory to create a connection
  - *Use security specified in name space by the JMS administrator for connection authentication*
    - *Connection connection = connectionFactory.createConnection();*
  - *Use application provided user ID and password for connection authentication (not supported in CICS)*
    - *Connection connection = connectionFactory.createConnection(userid,password);*
- *Start the connection*
  - *connection.start();*

All the information needed to connect to a queue manager

Start a session with the Queue Manager

- Use the connection object to create a session with the queue manager
  - Session object is the 'anchor' object used to work with other resources.
  - Session / QueueSession / TopicSession
  - *Session thisSession = connection.createSession();*
  - *Session thisSession = connection.createSession(Transacted, Acknowledge_Mode);*
  - Transacted attribute - true / false
  - Acknowledge_Mode :
    - AUTO_ACKNOWLEDGE
    - DUPS_OK_ACKNOWLEDGE
    - CLIENT_ACKNOWLEDGE
    - SESSION_TRANSACTED

Think of a session object as providing the function of an MQI connection handle

IBM MQ for z/OS Wildfire Workshop

Sample of Equivalent MQCONNX COBOL code

```
MOVE ' QML1' TO MQ-QMGR-NAME
COMPUTE MQCNO-VERSION = MQCNO-VERSION-5
COMPUTE MQCSP-AUTHENTICATIONTYPE = MQCSP-AUTH-USER-ID-AND-PWD

MOVE 'USERID' TO WS-USERID
MOVE ' PASSWORD ' TO WS-PASSWORD
COMPUTE MQCSP-CSPUSERIDLENGTH = 6
COMPUTE MQCSP-CSPPASSWORDLENGTH = 8.
SET MQCSP-CSPUSERIDPTR TO ADDRESS OF WS-USERID
SET MQCSP-CSPPASSWORDPTR TO ADDRESS OF WS-PASSWORD.
SET MQCNO-SECURITYPARMSPTR TO ADDRESS OF MQ-CSP

CALL 'MQCONNX'   USING MQ-QMGR-NAME
                 MQ-CNO
                 MQ-HCONN
                 MQ-COMPCODE
                 MQ-REASON .
* CHECK completion and reason codes
```

ConnectionFactory connectionFactory = (ConnectionFactory) context.lookup('jms/qmgr');
Connection connection = connectionFactory.createConnection(userid,password);
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
session.open();

Creating JMS Destination Objects

- Perform a JNDI lookup of a destination (queue) factory
  - *Destination destination = (Destination) context.lookup('jms/queue');*
- The destination's instance object is populated with information from the name space, such as:
  - *Base queue name*
  - *Properties (persisted/nonpresisted, read ahead allowed, etc.)*

e.g, all the information needed to access a queue

- Use both the destination and session instanace objects to create either producer (e.g. MQPUT) or consumer (e.g. MQGET) objects
  - *MessageProducer producer = session.createProducer(destination);*
  - *MessageSender consumer = session.createConsumer(destination);*

Think of a destination object as providing the function of an MQI queue handle

JMS *Producer* and *Consumer* objects methods

These objects provide several methods for interacting with a *destination,* a subset of the more common methods are shown below

- A subset of the methods available to *message producer* objects
  - setPriority(int priority);
  - send(Destination destination, Message message);
  - send(Message message);
  - close();
- A subset of the methods available to *message consumer* objects
  - receive();
  - receive(long timeout);
  - receiveNoWait();
  - close();

Sample MQOPEN QUEUE COBOL Sample

```
MOVE 'SYSTEM.DEFAULT.LOCAL.QUEUE'  TO MQOD-OBJECTNAME
MOVE MQOT-Q  TO MQOD-OBJECTTYPE
COMPUTE MQ-OBJ-OPTS = MQOO-OUTPUT + MQOO-PASS-ALL-CONTEXT

CALL 'MQOPEN' USING  MQ-HCONN,
                MQ-OBJ-DESC,
                MQ-OBJ-OPTS,
                MQ-OBJHAND,
                MQ-COMPCODE,
                MQ-REASON
```

MessageProducer producer = session.createProducer(destination);

Note that some open options specified in the MQOPEN COBOL API call are for JMS are provided as extended properties when the JNDI JMS destination object is created or by using methods.

Obtaining a JMS Topic Object

- Perform a JNDI lookup of a topic factory
  - *Topic topic = (Topic) context.lookup("jms/topic");*

All the information needed to access a topic

- Use both the topic and session objects to create either publisher (e.g. MQPUT) or subscriber (e.g. MQSUB) objects
  - *TopicPublisher publisher = sessionTopic.createPublisher(topic);*
  - *TopicSubscriber subscriber = sessionTopic.createSubscriber(topic);*
  - *TopicSubscriber durableSubscriber = session.CreateDurableSubscriber(topic,"Sub_name");*

Think of a topic object as providing the function of an MQI topic handle

JMS *publisher* and *subscriber* objects methods

These objects provide several methods for interacting with topics, the more interesting ones are below

- A subset of the methods available to *publisher* objects
  - setPriority(int priority);
  - publish(Message message);
  - getTopic();
  - close();
- A subset of the methods available to *subscriber* objects
  - receive();
  - getTopic();
  - close();

IBM MQ for z/OS Wildfire Workshop

JMS Message Types

Use the session object to create message objects

- BytesMessage : Unformatted binary data
  - *session.createBytesMessage(new byte[]);*
- TextMessage : Character data
  - *session.createTextMessage("String data");*
- StreamMessage : Sequence of typed data fields
  - *session.createStreamMessage();*
- MapMessage : Collection of typed data fields
  - *session.createMapMessage();*
- ObjectMessage : Serialized Java Object
  - *session.createObjectMessage();*

Working with JMS Message Object

- Use the session object to create a text message object
  - *message = session.createTextMessage(……);*
- Put the message to the destination (queue) using the *send* method
  - *producer.send(message);*
- Get a message from the destination (queue) using the *receive* method
  - *consumer.receive(message);*

COBOL Samples of MQPUT and MQGET

```
MOVE MQ-HMSG TO MQPMO-ORIGINALMSGHANDLE.
COMPUTE MQPMO-ACTION = MQACTP-NEW
COMPUTE MQ-PUT-BUFFLEN = L2.                    .

CALL 'MQPUT' USING MQ-HCONN
            MQ-OBJHAND
            MQ-MSG-DESC
            MQ-PUT-MSG-OPTS
            MQ-PUT-BUFFLEN
            WS-MQ-MESSAGE
            MQ-COMPCODE
            MQ-REASON.
```

producer.send(message);

consumer.receive(message);

```
MOVE LOW-VALUES TO MQMD-MSGID
            MQMD-CORRELID
MOVE SPACES    TO W02-COMMAND-REPLY
*
CALL 'MQGET' USING VD3-HCONN
            VD3-HOBJ
            MQMD
            MQGMO
            W02-REPLY-LENGTH
            W02-COMMAND-REPLY
            W00-DATA-LENGTH
            W03-COMPCODE
            W03-REASON.
```

Key Object Oriented programing point - Polymorphism: JMS method signature (*send* or *receive*) are the same regardless of the message type.

COBOL Return Code Checking

```
*
      CALL 'MQPUT1' USING VD3-HCONN
                  MQOD
                  MQMD
                  MQPMO
                  W02-DEFINE-LENGTH
                  W02-DEFINE-COMMAND
                  W03-COMPCODE
                  W03-REASON.
 *
      IF (W03-COMPCODE NOT = MQCC-OK) THEN
        MOVE 'DEFQ PUT1'   TO VD0-MSG1-TYPE
        MOVE W03-COMPCODE  TO VD0-MSG1-COMPCODE
        MOVE W03-REASON    TO VD0-MSG1-REASON
        MOVE VD0-MESSAGE-1 TO VD3-MSG
        GO TO CREATE-MAIL-QUEUE-TEMPQ-CLOSE
      END-IF.
```

IBM MQ for z/OS Wildfire Workshop

Java and JMS Exception Handling

- Java uses try/catch blocks

```
try {
    producer.send(message);
    …….
}  catch (JMSException jmsex)
    jmsex.printStackTrace();
}
```

JMS Message Selectors

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

Selector syntax

- Selectors can be:
  - ***Literals –*** ***"color = 'blue'"***
  - ***Byte strings -*** ***"myBytes = "0x0AFC23""***
  - ***Exact numeric literal -*** ***"NoItemsInStock > 20"***
  - ***Approximate numeric literal -*** ***"Difference < .7e+2"***
  - ***Boolean literals TRUE or FALSE -*** ***"AcctDetails = TRUE"***
  - ***Java identifiers –*** ***"JMSPriority >= 0"***
  - ***Expressions -*** ***"Type = 'car' AND color = 'blue' AND weight > 2500"***

White space is the same as it is defined for Java: space, horizontal tab, form feed, and line terminator.

Message Selectors

Provides a means for an application to request filtering of messages by the JMS provider based on message property

- Based on user message properties or header fields
  - *message.setStringProperty("Color", "Red");*
- Specified by message consumer
  - *consumer = session.createConsumer(destination, "Color = Red");*
  - *consumer = session.createConsumer(destination, "Type = 'car' AND color = 'blue' AND weight > 2500");*
  -

IBM MQ for z/OS Wildfire Workshop

Message Properties COBOL Sample

```
      COMPUTE MQSMPO-OPTIONS = MQSMPO-SET-FIRST.

*** SET PROPERTY DESCRIPTION (MQ-PROP-DESC)
     COMPUTE MQPD-OPTIONS = MQPD-NONE
     COMPUTE MQPD-SUPPORT = MQPD-SUPPORT-OPTIONAL
     COMPUTE MQPD-COPYOPTIONS = MQCOPY-DEFAULT
     COMPUTE MQPD-CONTEXT = MQPD-NO-CONTEXT

*** SET PROPERTY TYPE (MQ-PROP-TYPE)
     COMPUTE MQ-PROP-TYPE = MQTYPE-STRING

*** SET PROPERTY NAME (MQ-PROP-NAME)
     MOVE 'COLOR'            TO WS-PPTY-NAME.
     SET MQCHARV-VSPTR TO ADDRESS OF WS-PPTY-NAME.
     COMPUTE MQCHARV-VSLENGTH      = 5.
*** SET PROPERTY VALUE LENGTH (MQ-PROP-VALUE)
      MOVE 'RED'         TO MQ-PROP-VALUE.
      COMPUTE MQ-PROP-VAL-LENGTH = 3

      CALL 'MQSETMP' USING MQ-HCONN
                 MQ-HMSG
                 MQ-SET-MSG-PROP-OPTS
                 MQ-PROP-NAME
                 MQ-PROP-DESC
                 MQ-PROP-TYPE
                 MQ-PROP-VAL-LENGTH
                 MQ-PROP-VALUE
                 MQ-COMPCODE
                 MQ-REASON.
```

message.setStringProperty("Color", "Red");

JMS Properties v MQ Properties

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

Message Headers and Properties

- JMS header fields with corresponding MQMD fields

JMS header field	MQMD field
JMSCorrelationID	CorrelId
JMSDeliveryMode	Persistence
JMSExpiration	Expiry
JMSPriority	Priorty
JMSMessageID	MsgID
JMSTimeStamp	PutDate/PutTime

message.setJMSCorrelationID(string);
message.setJMSPriority(integer);

Message Headers and Properties

- JMS properties mapping to MQMD fields

JMS property	MQMD field
JMSXUserid	UserIdentifier
JMSXAppID	PutApplName
JMSXDeliveryCount	BackoutCount
JMSXGroupID	GroupID
JMSXGroupSeq	MsgSeqNumber

message.setStringProperty("JMSXGROUPID","00000001");
message.setIntProperty("JMSXGroupSeq",1);

These properties are normally set the provider.

IBM MQ for z/OS Wildfire Workshop

Message Headers and Properties

- Outgoing message field mapping

JMS header field name	MQMD field userd for transmission
JMSDestination	
JMSDeliveryMode	Persistence
JMSExpiration	Expiry
JMSPriority	Priority
JMSMessgeID	MsgID
JMSTimeStamp	PutDate/PutTime
JMSCorrelationID	CorrelId
JMSReplyTo	ReplyToQ/ReplyToQMgr
JMSType	
JMSRedelivered	

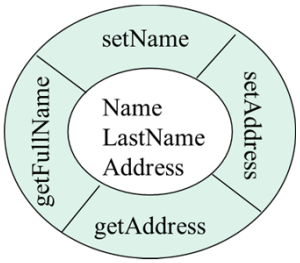
message.setJMSReply(queue);
message.setJMSExpiration(long);

Developing IBM MQ JMS Applications

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

Creating Java Object from COBOL Copy Books

```
01 Customer
    10 Name        PIC X(20)
    10 LastName    PIC X(20)
    10 Address     PIC X(40).
```

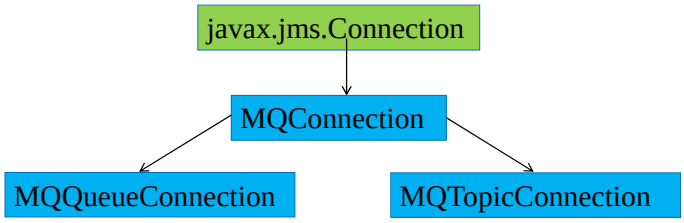


You may need to working with individual fields in a message. There are tools available to create Java objects from COBOL copy books.

- The J2C component of the Java EE Connectors feature of *IBM Rational Application Developer*
- The *JZOS Assembler/COBOL Record Generator* utility included as part of the *IBM Experimental JZOS Batch Toolkit for z/OS SDKs*

Inheritance and Extensions

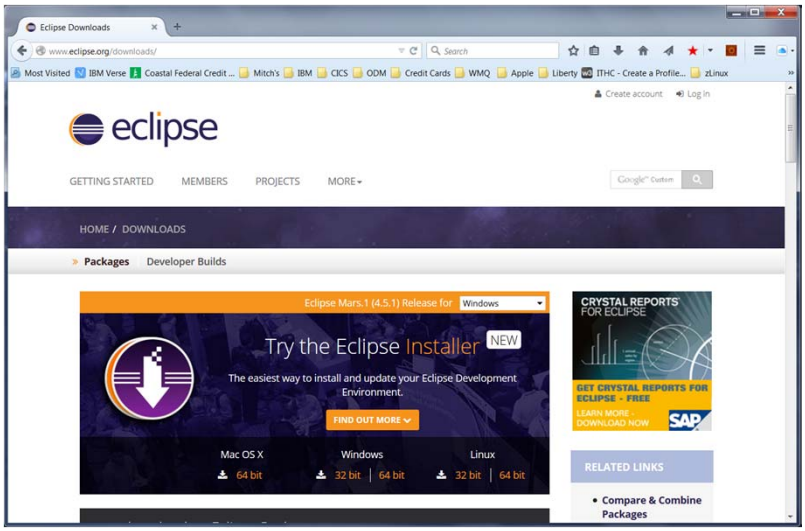
- The JMS Java classes provided by IBM MQ are extensions of the base JMS classes in Java package javax.jms
  - Extensions augment the functionally provided in the base class
- For example, the IBM MQ JMS class MQConnection in Java package com.ibm.mq.jms extends javax.jms.Connection



- Public methods of classes higher in the hierarchy are available to subordinated classes
- *JMS Provider portability is lost once an extended class is used*

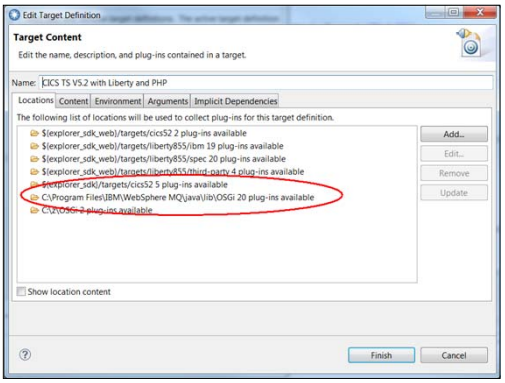
IBM MQ for z/OS Wildfire Workshop

Download an Eclipse Development Tool

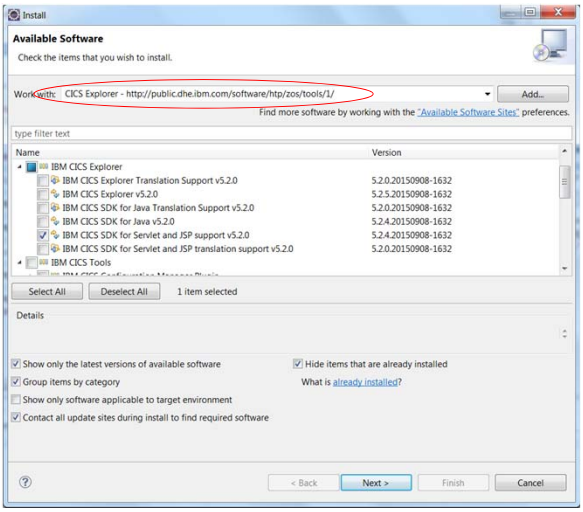


Developing and Deploying CICS JMS Applications

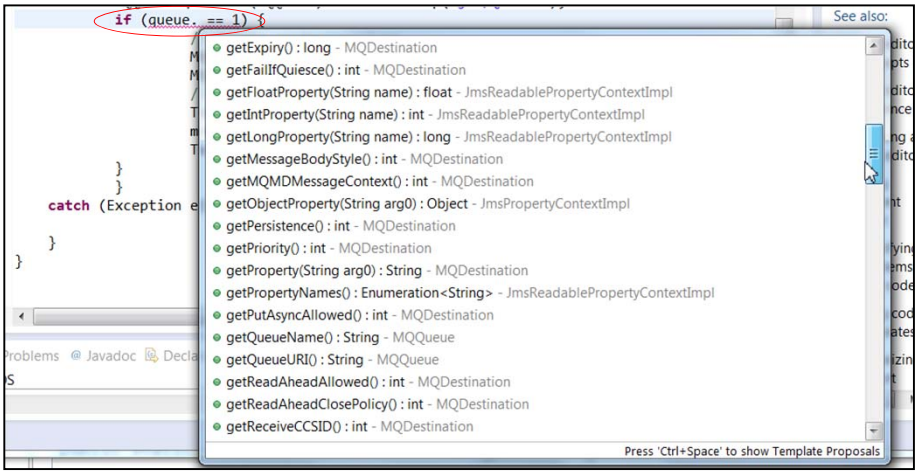
- Development of CICS JMS applications requires the use of the CICS SDK Eclipse tool and that it be configured with the IBM MQ OSGi plug-ins (Fix Pack 2).



Extend Eclipse by adding the CICS SDK

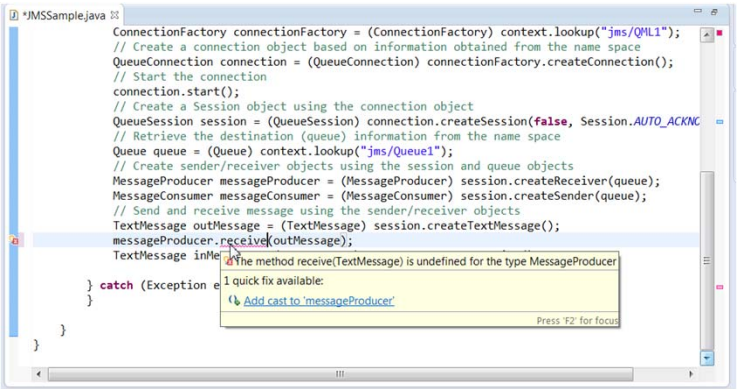


Code Assist Feature in Eclipse



The Eclipse IDE can be downloaded from https://eclipse.org/downloads/

Code and Syntax Checking - Eclipse



Sample Code using IBM MQ JMS Classes

```
// Obtain a connection factory from the name space
MQConnectionFactory connectionFactory = (MQConnectionFactory) context.lookup("jms/qmgr");
// Create a connection object using the ConnectionFactory object
MQQueueConnection connection = (MQQueueConnection) connectionFactory.createConnection();
// Start the connection
connection.start();
// Obtain a session to the queue manager using the Connection object
MQQueueSession session = (MQQueueSession) session = connection.createSession(transacted,acknowlegeMode
/ Retrieve the queue information from the name space
MQQueue queue = (MQQueue)context.lookup("jms/queue");
// Check extended attribute
if (queue.getFailIfQuiesce() == WMQConstants.WMQ_FIQ_YES {
        // Create sender/receiver objects
        MQMessageProducer messageProducer = (MQMessageProducer) session.createReceiver(queue);
        MQMessageConsumer messageConsumer = (MQMessageConsumer) session.createSender(queue);
        // Send and receive message
        TextMessage outMessage = (TextMessage) session.createTextMessage();
        messageProducer.send(outMessage);
        TextMessage inMessage = (TextMessage)  messageConsumer.receive();
}
```


Combining base JMS classes and IBM MQ Java classes

*CICSBridgeResults.java

```
        // Create a request message instance variable and set its JMS MQ properties
        producer = session.createProducer(requestDestination);
        requestMessage = (JMSBytesMessage) session.createBytesMessage();
        requestMessage.setIntProperty("JMS_IBM_MsgType",MQConstants.MQMT_REQUEST);  // Request/reply
        requestMessage.setIntProperty("JMS_IBM_Encoding",MQConstants.MQENC_S390);   // S390 encoding (785)
        requestMessage.setStringProperty("JMS_IBM_Format", MQConstants.MQFMT_CICS); // MQCIH + COMMAREA
        requestMessage.setJMSCorrelationIDAsBytes(MQConstants.MQCI_NEW_SESSION);    // Start a new session
        requestMessage.setIntProperty("WMQ_MESSAGE_BODY",MQConstants.WMQ_MESSAGE_BODY_MQ); // Do not include a MQRFH2 header
        requestMessage.setJMSReplyTo(responseDestination);                          // Set the response queue name

        // Set the MQCIH header attributes
        if (sc.getAttribute("cics.password") != null) {
           mqcih.setAuthenticator(((String) sc.getAttribute("cics.password")).toUpperCase());
        }

        mqcih.setFormat(MQConstants.MQFMT_NONE);
        mqcih.setADSDescriptor(MQConstants.MQCADSD_NONE);
        mqcih.setLinkType(MQConstants.MQCLT_PROGRAM);
        mqcih.setOutputDataLength(mqcihSize + programNameSize + cicsRequest.getSize());
        mqcih.setReplyToFormat(MQConstants.MQFMT_NONE);
        mqcih.setTransactionId("ADS2");
        mqcih.setUOWControl(MQConstants.MQCUOWC_ONLY);
        mqcih.setVersion(MQConstants.MQCIH_VERSION_2);

        // Add the MQCIH header to beginning of the message
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        mqcih.write (new DataOutputStream (out),MQConstants.MQENC_NATIVE,819);
        byte[] bytes = out.toByteArray();
        requestMessage.writeBytes(bytes);

        // Append the target CICS program name to the message
        requestMessage.writeBytes(programName.getBytes("IBM-1047"));  // Set to EBCDIC
        // Append the COMMAREA to the message
        requestMessage.writeBytes(cicsRequest.getBytes());

        // Send (PUT) the message on the request queue
        producer.send(requestMessage);
```

Configuring JNDI Namespaces on z/OS

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

What is Java Naming and Directory Interface (JNDI)?

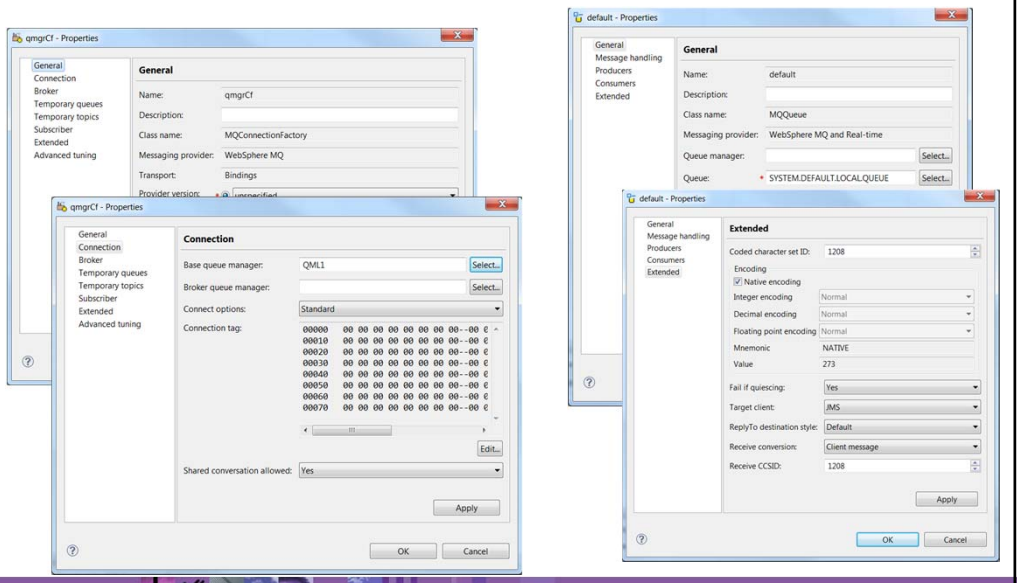
- *ConnectionFactory connectionFactory = (ConnectionFactory) context.lookup('jms/qmgr');*
- *Destination destination = (Destination) context.lookup('jms/queue");*
- A JNDI service provides common naming and directory services to Java clients so they can look up or obtain information simply by specifying a name (a JNDI name) of a resource.
- Java Naming and Directory Interface
  - LDAP (seldom, if ever, used on z/OS)
  - File system

Configuring JMS JNDI Information on z/OS

- For CICS use either
  - MQ Explorer
    - Upload the configuration file to OMVS in binary format
  - JMSAdmin, an OMVS command
    - Found in /usr/lpp/mqm/V8R0M0/java/bin
    - *./JMSAdmin –cfg configuration.file*
- For Liberty, manually update the ***server.xml*** file
  - Windows Liberty provides a nice set of tools to create the necessary configuration stanzas

IBM MQ for z/OS Wildfire Workshop

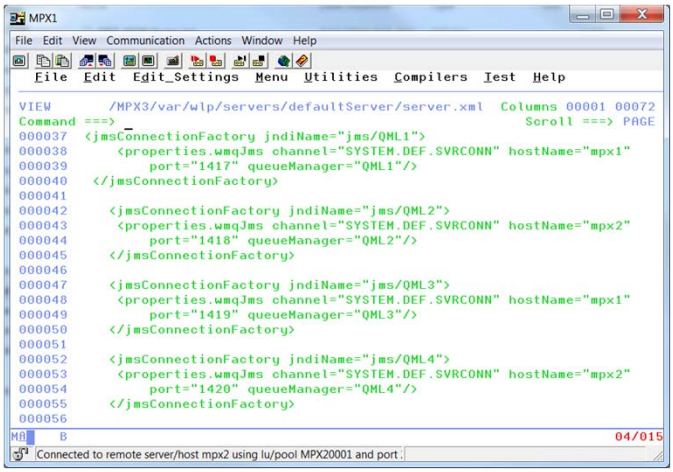
JMS Configuration Wizards in MQ Explorer



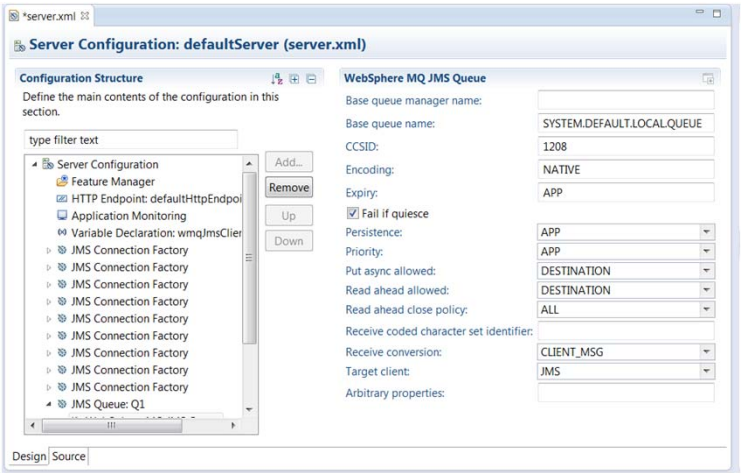
JMSAdmin command example

```
JMSAdmin -cfg myJNDI.config

Licensed Materials - Property of IBM
5724-H72, 5655-R36, 5724-L26, 5655-L82
(c) Copyright IBM Corp. 2008, 2011 All Rights Reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
Starting WebSphere MQ classes for Java(tm) Message Service Administrati

InitCtx> dis ctx

  JMSADM4089 InitCtx

      .bindings                 java.io.File
      myJNDI.config             java.io.File
   a  teamxx                    com.ibm.mq.jms.MQQueue
   a  default                   com.ibm.mq.jms.MQQueue
   a  qmgrCf                    com.ibm.mq.jms.MQConnectionFactory

  5 Object(s)
    0 Context(s)
    5 Binding(s), 3 Administered

InitCtx>
```


Liberty JMS JNDI Configuration



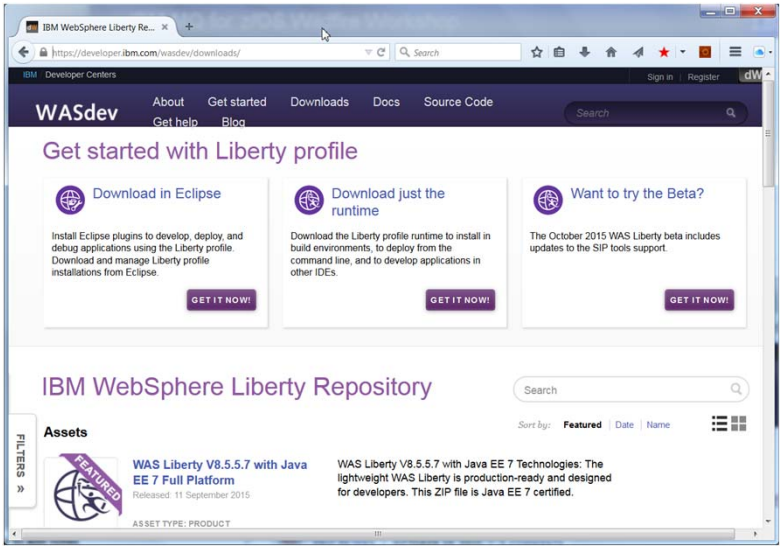
Liberty JMS Configuration Editor



Liberty JMS Configuration Source

```
*server.xml
<jmsConnectionFactory jndiName="jms/MQS1">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="wg31" port="1414" queueManager="MQS1"/>
</jmsConnectionFactory>

<jmsConnectionFactory jndiName="jms/QMZB">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="mpx3" port="1416" queueManager="QMZB"/>
</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>

<jmsConnectionFactory jndiName="jms/WMQ8">
    <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostName="localzos" port="1436" queueManager="WMQ8"/>
</jmsConnectionFactory>

<jmsQueue id="Q1" jndiName="jms/Q1">
    <properties.wmqJms baseQueueName="SYSTEM.DEFAULT.LOCAL.QUEUE" failIfQuiesce="true"/>
</jmsQueue>
Design Source
```

Download a Liberty profile runtime



Integrating JMS in CICS and Liberty

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

IBM MQ JMS and CICS

- IBM MQ JMS support added in the service stream
  - CICS APAR
    - For V5.2 PI32151
  - MQ APARs
    - For V7.1: JMS PI29770 (supercedes 7.1.0.6) or later CSD
    - For V8: JMS 8.0.0.2 or later CSD + MQ base PI28482
- CICS only support JNDI configuration managed in an OMVS file
- CICS does not support
  - JMS listeners
  - Providing User IDs and passwords when creating connections
- Logical unit of work will be controlled by CICS unless the Session or JMSContext were created using the Session.AUTO_ACKNOWNLEDGE or JMSContext.AUTO_ACKNOWLEDGE flag.

CICS JMS Enablement

- The application identifies the location of JNDI configuration file

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

- The CICS system programmer updates the OSGI_BUNDLES property in the CICS region's JVMServer profile to include the IBM MQ JMS supplied OSGi jar files.

N.B. OSGi (Open Service Gateway initiative) framework for deploying and administering Java applications. The OSGi framework restructures the components of an application as individual bundles of components or packages that are loosely coupled but when combined constitute an application.

CICS JMS JVMServer Enablement

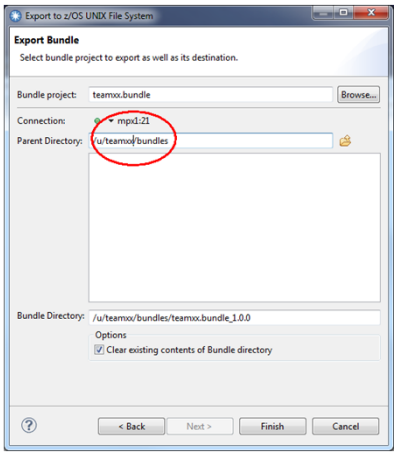
- Below is an example of the required updates and other relevant configuration variables.

```
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]
```

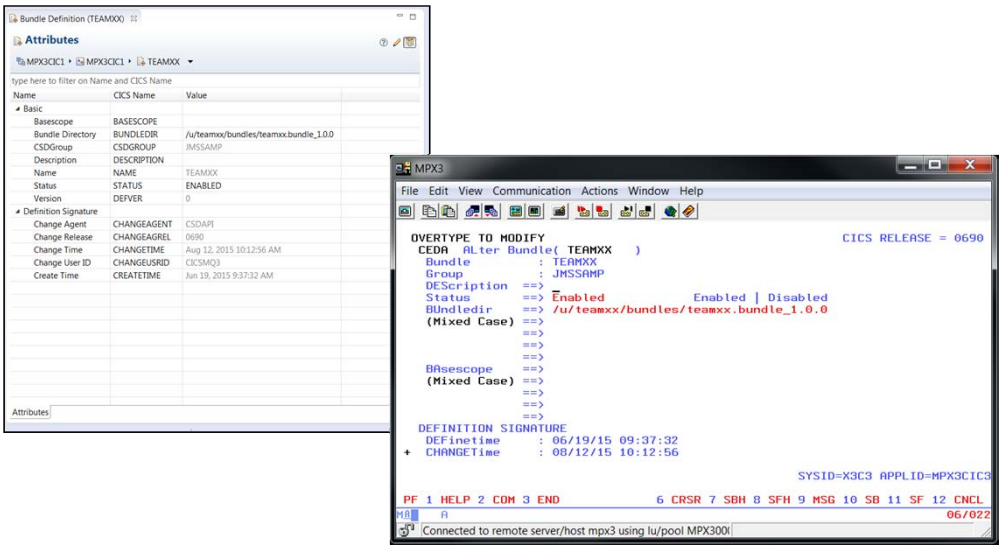
1. Identifies the OMVS directory where this JVMServer will use for configuration files, logs, error messages, etc.
2. Identifies the OSGi Jar file bundles required for the CICS and MQ JMS Java application.
3. Identifies the standard Java output (STDOUT) file (within WORK_DIR)
4. Identifies the standard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

Deploying the CICS JMS Applications

- Once developed, the application bundle is deployed to the OMVS HFS directory defined in the CICS bundle resource



Required CICS Bundle Definition



Snippet of stack trace output

```
June 22, 2015 3:35:53 PM GMT[JMSSAMP.TASK244.T0XX] com.ibm.msg.client.wmq.internal.WMQConnection
Exception ignored as no exception listener is registered: '
                Message : com.ibm.msg.client.jms.DetailedIllegalStateException: JMSWMQ1107: A problem with this connection has occurred.
An error has occurred with the WebSphere MQ JMS connection.
Use the linked exception to determine the cause of this error.
                 Class : class com.ibm.msg.client.jms.DetailedIllegalStateException
                 ……..
Caused by [1] --> Message : com.ibm.mq.MQException: JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED')
reason '2033' ('MQRC_NO_MSG_AVAILABLE').
                 Class : class com.ibm.mq.MQException
                 Stack : com.ibm.msg.client.wmq.common.internal.Reason.createException(Reason.java:202)
                     : com.ibm.msg.client.wmq.internal.WMQMessageConsumer.checkJmqiCallSuccess(WMQMessageConsumer.java:124)
                     : com.ibm.msg.client.wmq.internal.WMQConsumerShadow.getMsg(WMQConsumerShadow.java:1810)
                     : com.ibm.msg.client.wmq.internal.WMQSyncConsumerShadow.receiveInternal(WMQSyncConsumerShadow.java:230)
                     : com.ibm.msg.client.wmq.internal.WMQConsumerShadow.receive(WMQConsumerShadow.java:1446)
                     : com.ibm.msg.client.wmq.internal.WMQMessageConsumer.receive(WMQMessageConsumer.java:533)
                     : com.ibm.msg.client.jms.internal.JmsMessageConsumerImpl.receiveInboundMessage(JmsMessageConsumerImpl.java:1015)
                     : com.ibm.msg.client.jms.internal.JmsMessageConsumerImpl.receive(JmsMessageConsumerImpl.java:652)
                     : com.ibm.mq.jms.MQMessageConsumer.receive(MQMessageConsumer.java:209)
                     : com.ibm.cicsjms.samples.JMSSample.main(JMSSample.java:143)
                     : sun.reflect.GeneratedMethodAccessor7.invoke(null:-1)
                     ……'.
```

This stack trace appeared in /var/wlp/cics/MPX3CIC1/mqjms.log.1. The location where this trace was written was determined by the **WORK_DIR** variable in the JVMServer profile.

Sample of JCICS code to handle exception

```
Try {
  …..
} catch (JMSException jmsex) {
        workContainer.setMsg(jmsex.getLocalizedMessage() + "</br>" +
                jmsex.getLinkedException().getLocalizedMessage() + "</br>");
        try {
                responseContainer = channel.createContainer("JMSSAMPResponse");
                responseContainer.put(workContainer.getBytes());
                task.rollback();            // EXEC CICS SYNCPOINT ROLLBACK
        } catch (Exception e) {
        e.printStackTrace();
}
```

IBM MQ for z/OS Wildfire Workshop

Liberty server.xml Messages and Logs

- Liberty messages and logs are written ASCII format and can be located in file */var/wlp/servers/defaultServer/logs/messages.log*

```
[9/27/15 14:24:00:121 GMT] 0000002f com.ibm.ws.logging.internal.impl.IncidentImpl I FFDC1015I: An FFDC Incident has been created: "com.ibm.mq.connector.DetailedResourceException:
MQJCA1011: Failed to
allocate a JMS connection., error code: MQJCA1011 An internal error caused an attempt to allocate a connection to fail. See the linked exception for details of the failure.
com.ibm.ejs.j2c.poolmanager.FreePool.
createManagedConnectionWithMCWrapper 199" at ffdc_15.09.27_14.24.00.0.log
[9/27/15 14:24:00:125 GMT] 0000002f SystemErr                    R com.ibm.msg.client.jms.DetailedJMSException: MQJCA1011: Failed to allocate a JMS connection.
An internal error caused an attempt to allocate a connection to fail.
See the linked exception for details of the failure.
[9/27/15 14:24:00:125 GMT] 0000002f SystemErr                    R      at com.ibm.mq.connector.services.JCAExceptionBuilder.buildException(JCAExceptionBuilder.java:146)
….
[9/27/15 14:24:00:130 GMT] 0000002f SystemErr                    R Caused by:
[9/27/15 14:24:00:131 GMT] 0000002f SystemErr                    R com.ibm.msg.client.jms.DetailedJMSSecurityException: JMSWMQ2013: The security authentication was not
valid that was supplied for  QueueManager 'QMZA' with connection mode 'Client' and host name 'mpx3(1414)'.
Please check if the supplied username and password are correct on the QueueManager to which you are connecting.
[9/27/15 14:24:00:131 GMT] 0000002f SystemErr                    R      at com.ibm.msg.client.wmq.common.internal.Reason.reasonToException(Reason.java:521)
….
[9/27/15 14:24:00:136 GMT] 0000002f SystemErr                    R Caused by:
[9/27/15 14:24:00:136 GMT] 0000002f SystemErr                    R com.ibm.mq.MQException: JMSCMQ0001: WebSphere MQ call failed with compcode '2'
('MQCC_FAILED') reason '2035' ('MQRC_NOT_AUTHORIZED').
[9/27/15 14:24:00:137 GMT] 0000002f SystemErr                    R      at com.ibm.msg.client.wmq.common.internal.Reason.createException(Reason.java:209)
[9/27/15 14:24:00:137 GMT] 0000002f SystemErr                    R      ... 36 more
```

- *The target directory for messages.log is determined by concatenating the **PATH** specified by DD name **WLPUDIR** in the server region's startup JCL, e.g. /var/wlp, with the subdirectory /servers/ with the name of the server, e.g. defaultServer, and the subdirectory /logs.*

Questions???