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

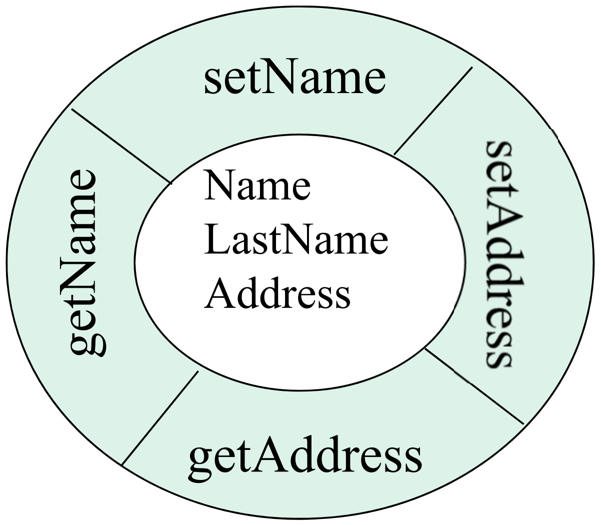
Basic Java Messing Service Programming

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).
```

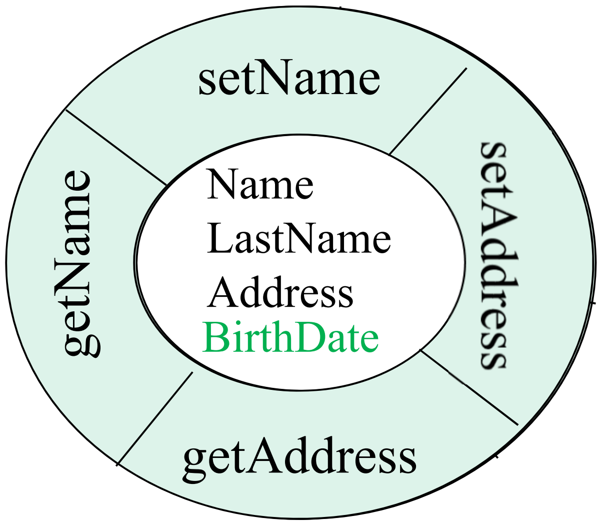
```
Customer customer = new Customer();
customer.setName("John");
newAddress = customer.getAddress();
```

```
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



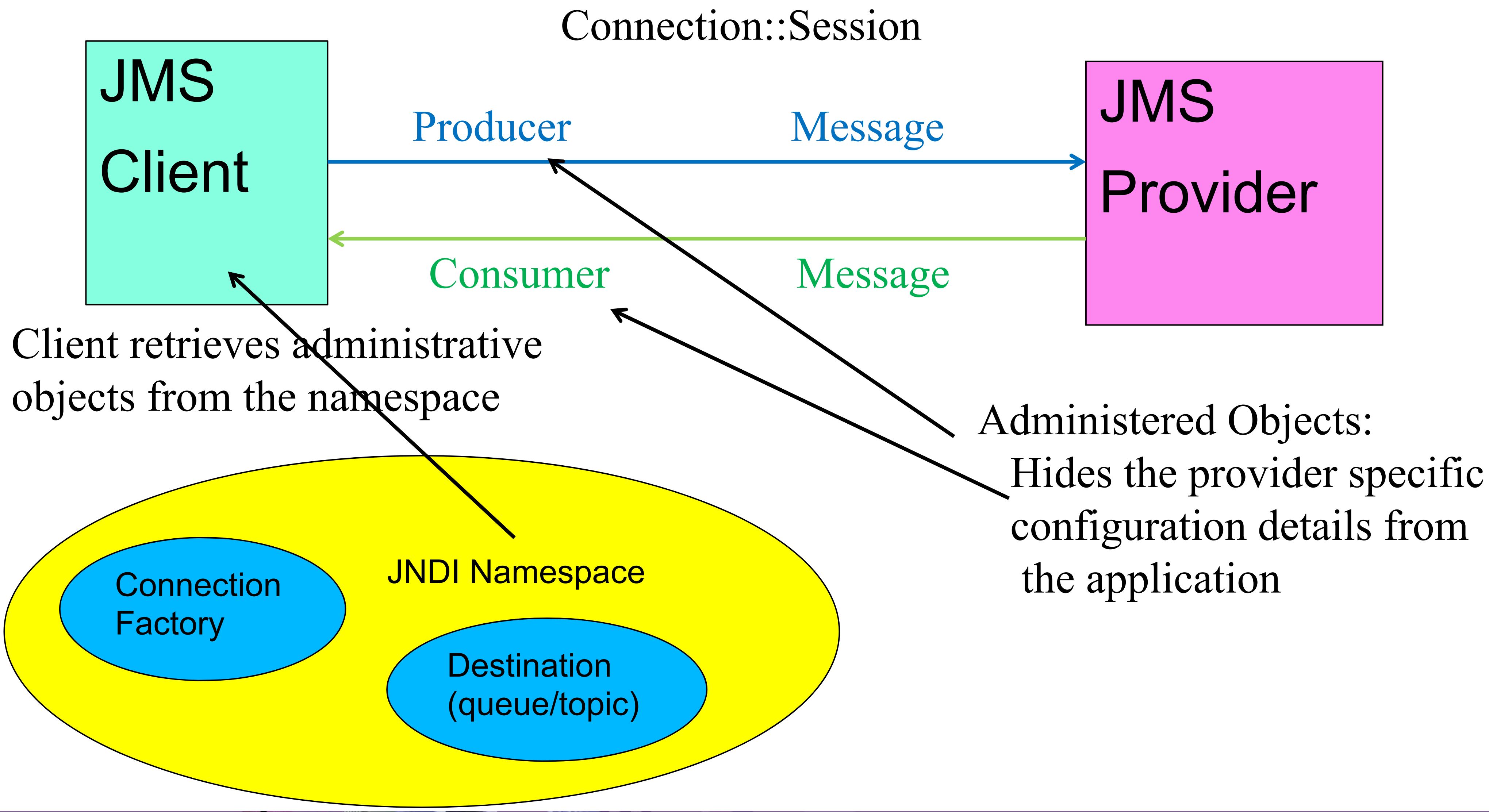
```
01 Customer
      10 Name         PIC X(20)
      10 LastName     PIC X(20)
      10 Address      PIC X(40).
      10 BirthDate    PIC X(10).
 01  FullName         PIC X(40).
```

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

```
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


Connection::Session
JMS Client
Producer
Message
JMS Provider
Consumer
Message
Client retrieves administrative objects from the namespace
Administered Objects:
Hides the provider specific configuration details from the application
JNDI Namespace
Connection Factory
Destination (queue/topic)

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

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();

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.
```

Java and JMS Exception Handling

- Java uses try/catch blocks

```
try {
     producer.send(message);
      …….
}  catch (JMSException jmsex)
    jmsex.printStackTrace();
} catch (Exception ex)
   ex.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“);*
  -

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.
```

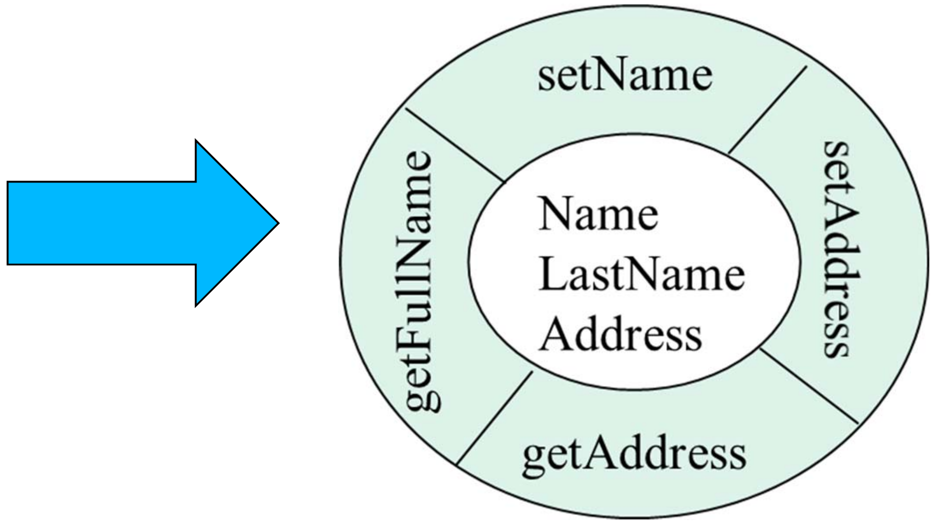
message.setStringProperty(“Color”, “Red”);

Developing IBM MQ JMS Applications

Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

Creating Java Object from COBOL Copy Books

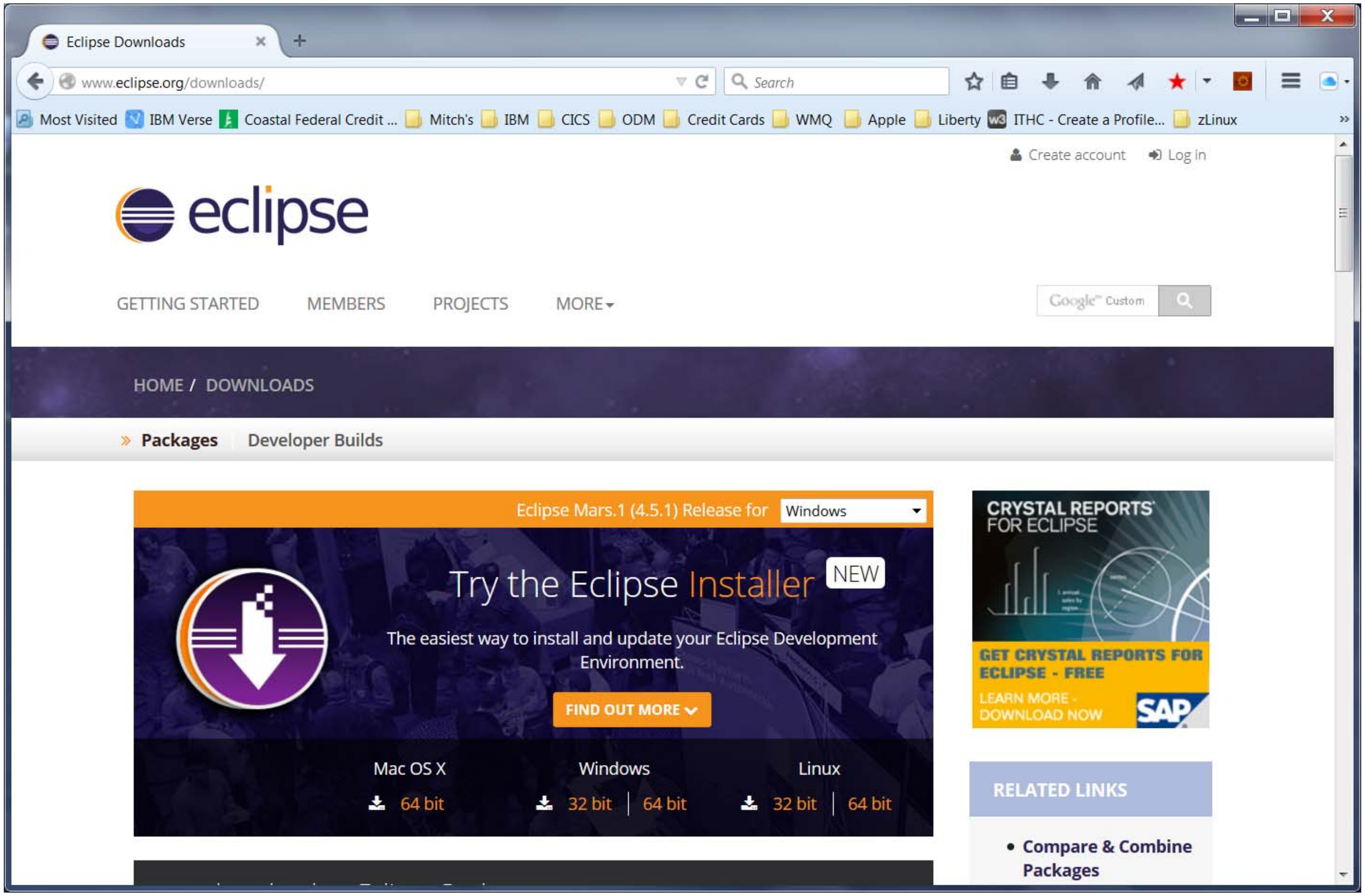
```
01 Customer
     10 Name        PIC X(20)
     10 LastName    PIC X(20)
     10 Address     PIC X(40).
```



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*

Download an Eclipse Development Tool


Eclipse Downloads

www.eclipse.org/downloads/

Most Visited | IBM Verse | Coastal Federal Credit ... | Mitch's | IBM | CICS | ODM | Credit Cards | WMQ | Apple | Liberty | ITHC - Create a Profile... | zLinux

Create account | Log in

eclipse

GETTING STARTED | MEMBERS | PROJECTS | MORE

HOME / DOWNLOADS

» Packages | Developer Builds

Eclipse Mars.1 (4.5.1) Release for Windows

Try the Eclipse Installer NEW

The easiest way to install and update your Eclipse Development Environment.

FIND OUT MORE

Mac OS X	Windows	Linux
64 bit	32 bit | 64 bit	32 bit | 64 bit

CRYSTAL REPORTS FOR ECLIPSE

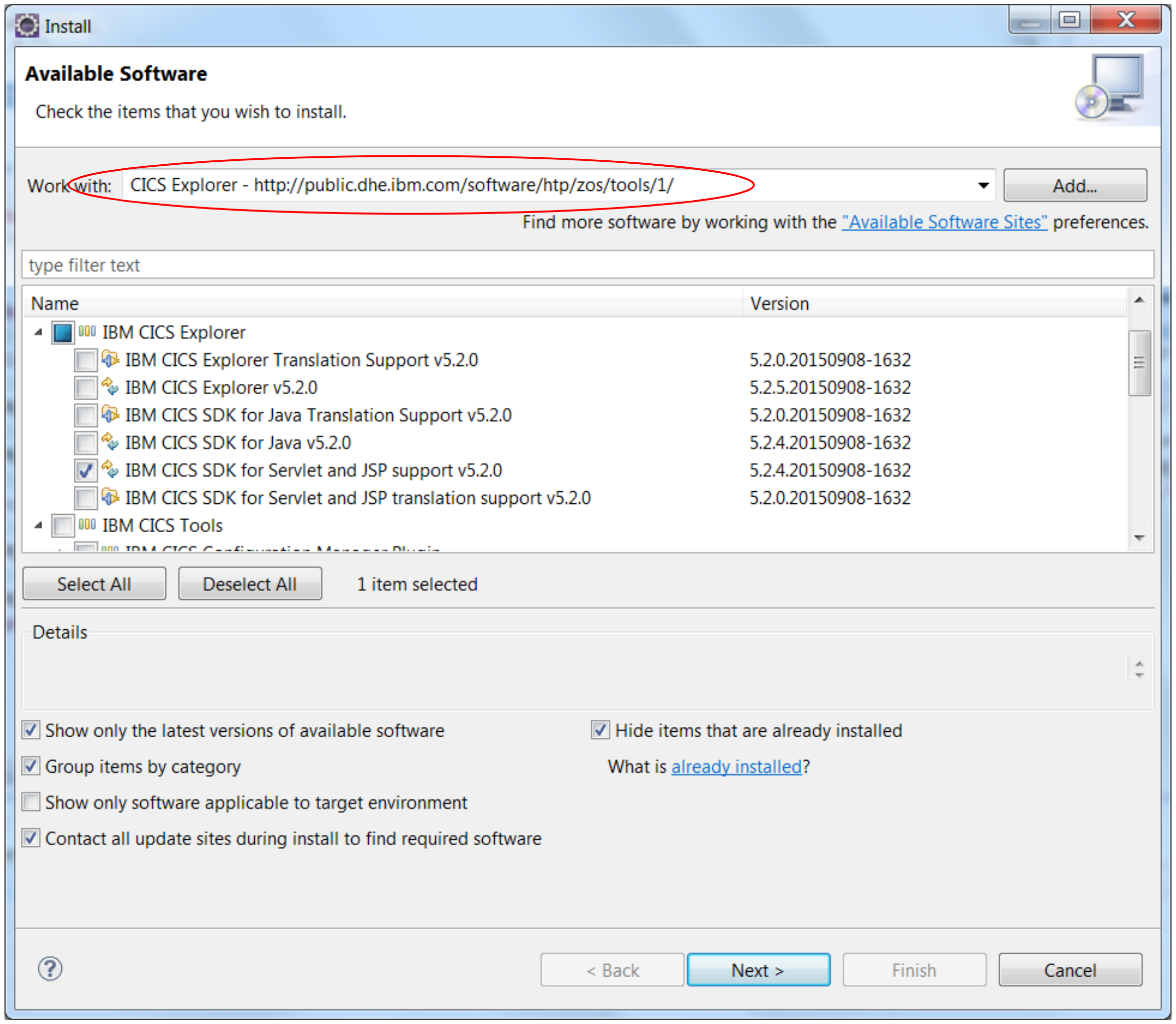
GET CRYSTAL REPORTS FOR ECLIPSE - FREE

LEARN MORE - DOWNLOAD NOW

RELATED LINKS

- Compare & Combine Packages

Extend Eclipse by adding the CICS SDK



Install

Available Software

Check the items that you wish to install.

Work with: CICS Explorer - http://public.dhe.ibm.com/software/htp/zos/tools/1/

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
▪ IBM CICS Explorer	
☐ IBM CICS Explorer Translation Support v5.2.0	5.2.0.20150908-1632
☐ IBM CICS Explorer v5.2.0	5.2.5.20150908-1632
☐ IBM CICS SDK for Java Translation Support v5.2.0	5.2.0.20150908-1632
☐ IBM CICS SDK for Java v5.2.0	5.2.4.20150908-1632
✓ IBM CICS SDK for Servlet and JSP support v5.2.0	5.2.4.20150908-1632
☐ IBM CICS SDK for Servlet and JSP translation support v5.2.0	5.2.0.20150908-1632
☐ IBM CICS Tools	

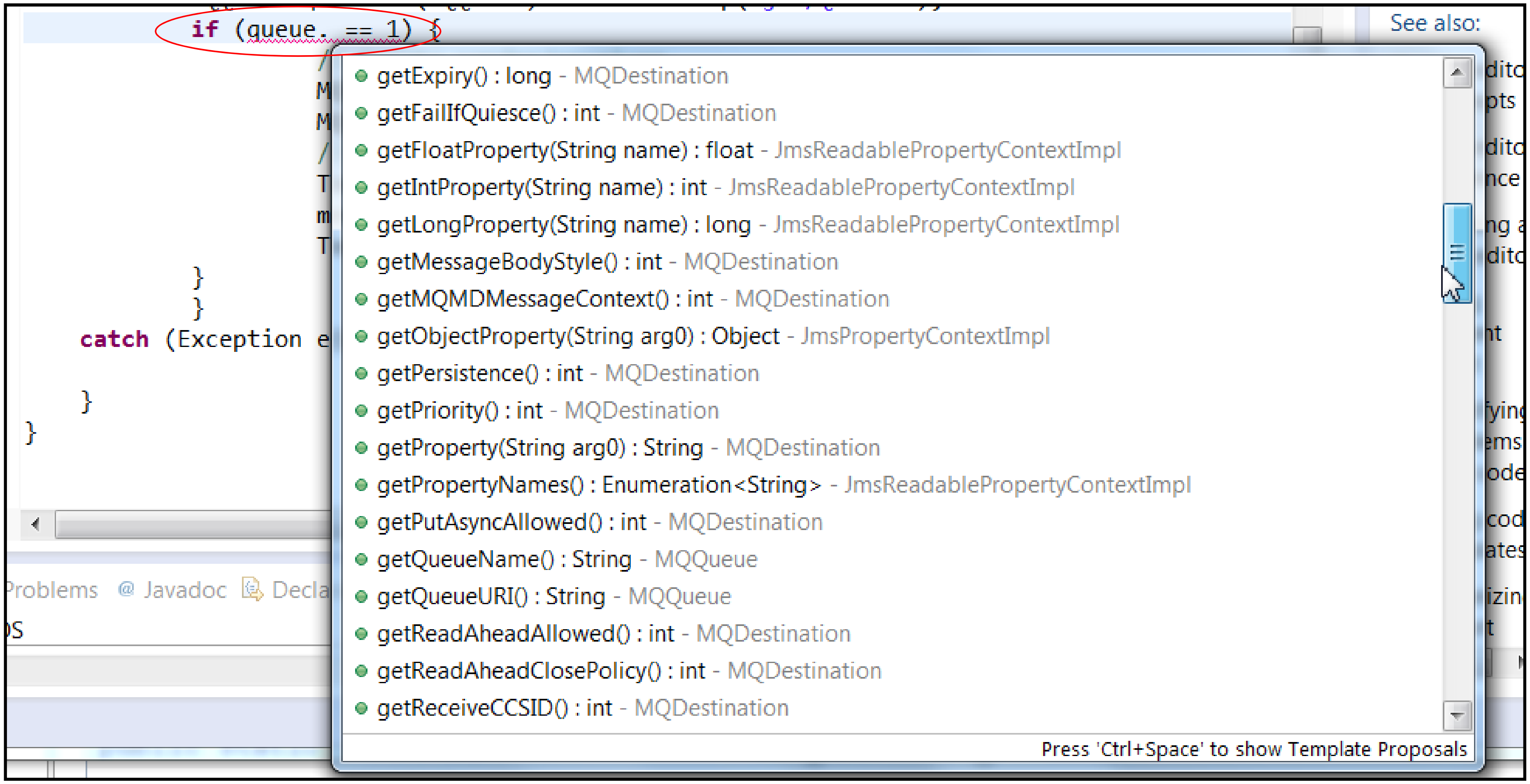
1 item selected

Details

- [✓] Show only the latest versions of available software
- [✓] Group items by category
- [ ] Show only software applicable to target environment
- [✓] Contact all update sites during install to find required software
- [✓] Hide items that are already installed

What is already installed?

Code Assist Feature in Eclipse



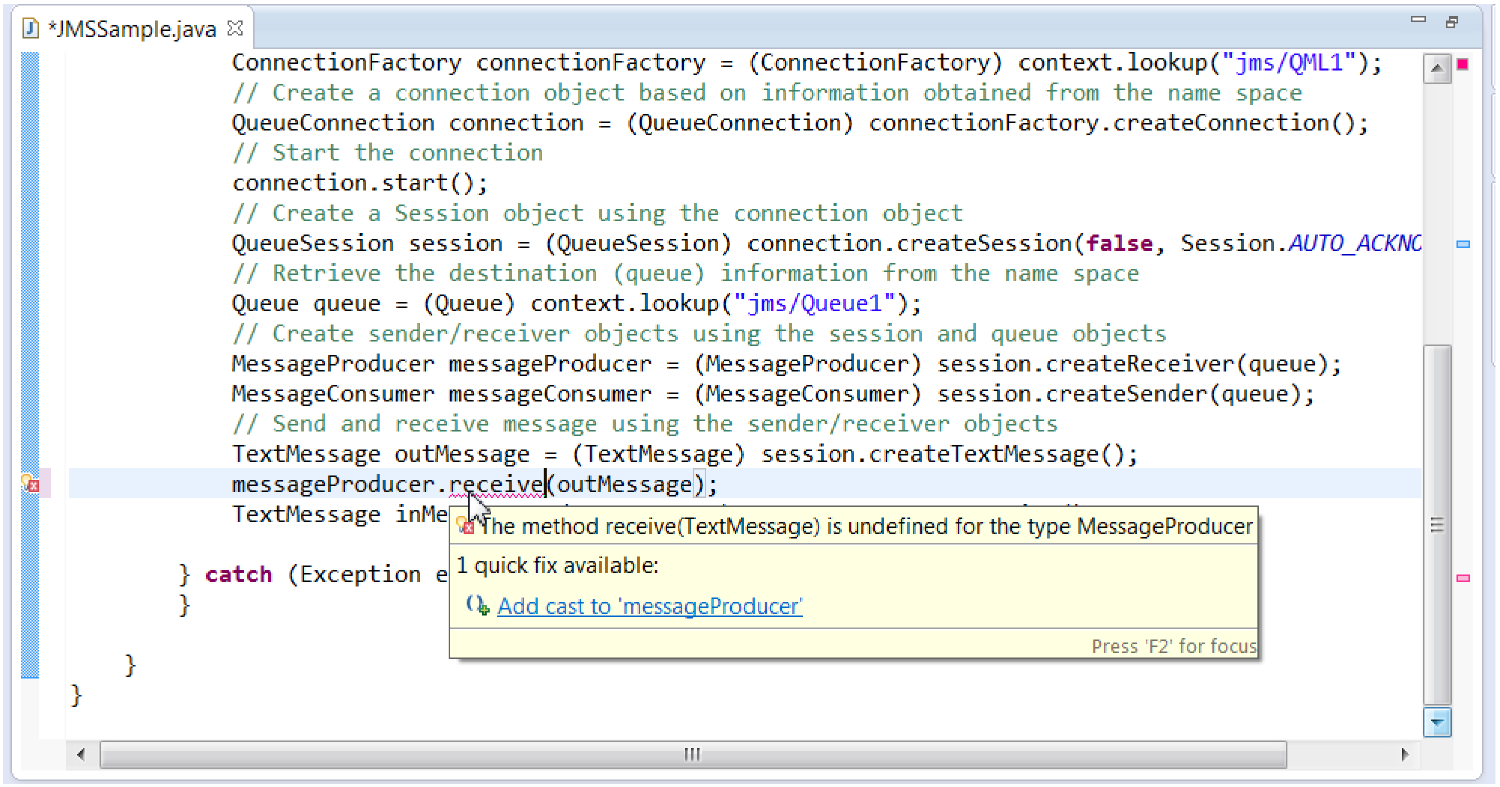
```
        if (queue. == 1) {
    catch (Exception e
    }
}
```

- getExpiry() : long - MQDestination
- getFailIfQuiesce() : int - MQDestination
- getFloatProperty(String name) : float - JmsReadablePropertyContextImpl
- getIntProperty(String name) : int - JmsReadablePropertyContextImpl
- getLongProperty(String name) : long - JmsReadablePropertyContextImpl
- getMessageBodyStyle() : int - MQDestination
- getMQMDMessageContext() : int - MQDestination
- getObjectProperty(String arg0) : Object - JmsPropertyContextImpl
- getPersistence() : int - MQDestination
- getPriority() : int - MQDestination
- getProperty(String arg0) : String - MQDestination
- getPropertyNames() : Enumeration<String> - JmsReadablePropertyContextImpl
- getPutAsyncAllowed() : int - MQDestination
- getQueueName() : String - MQQueue
- getQueueURI() : String - MQQueue
- getReadAheadAllowed() : int - MQDestination
- getReadAheadClosePolicy() : int - MQDestination
- getReceiveCCSID() : int - MQDestination

Press 'Ctrl+Space' to show Template Proposals

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

Code and Syntax Checking - Eclipse



```
            ConnectionFactory connectionFactory = (ConnectionFactory) context.lookup("jms/QML1");
            // Create a connection object based on information obtained from the name space
            QueueConnection connection = (QueueConnection) connectionFactory.createConnection();
            // Start the connection
            connection.start();
            // Create a Session object using the connection object
            QueueSession session = (QueueSession) connection.createSession(false, Session.AUTO_ACKNC
            // Retrieve the destination (queue) information from the name space
            Queue queue = (Queue) context.lookup("jms/Queue1");
            // Create sender/receiver objects using the session and queue objects
            MessageProducer messageProducer = (MessageProducer) session.createReceiver(queue);
            MessageConsumer messageConsumer = (MessageConsumer) session.createSender(queue);
            // Send and receive message using the sender/receiver objects
            TextMessage outMessage = (TextMessage) session.createTextMessage();
            messageProducer.receive(outMessage);
            TextMessage inMe

        } catch (Exception e
        }

    }
}
```

The method receive(TextMessage) is undefined for the type MessageProducer

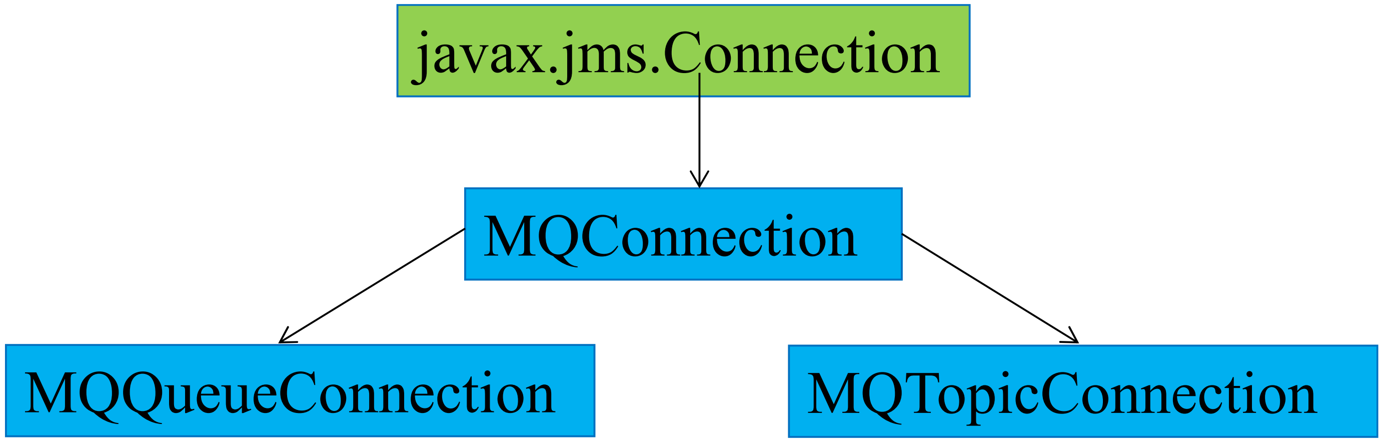
1 quick fix available:

Add cast to 'messageProducer'

Press 'F2' for focus

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

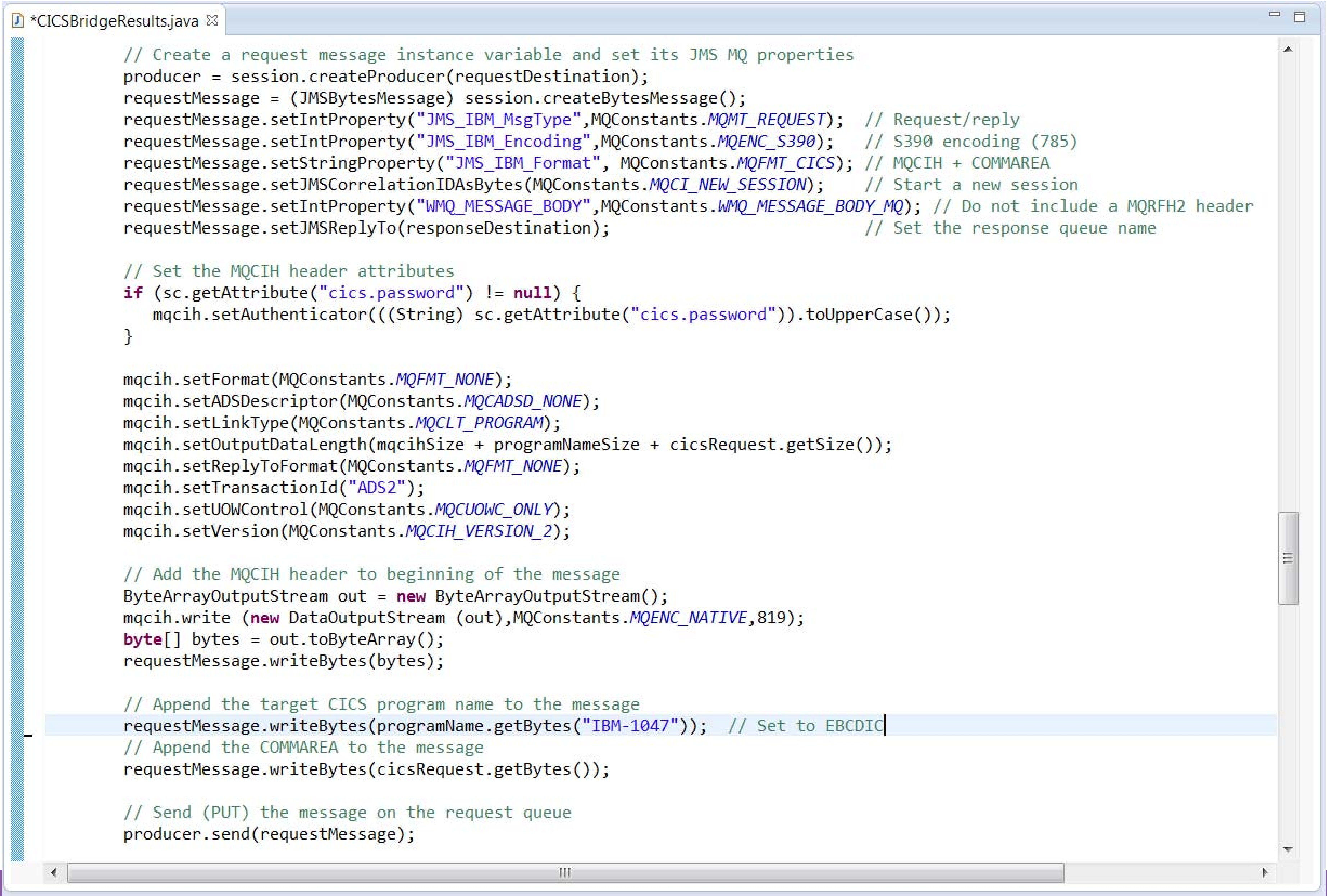


- Public methods of classes higher in the hierarchy are available to subordinated classes
- *JMS Provider portability is lost once an extended class is used*

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

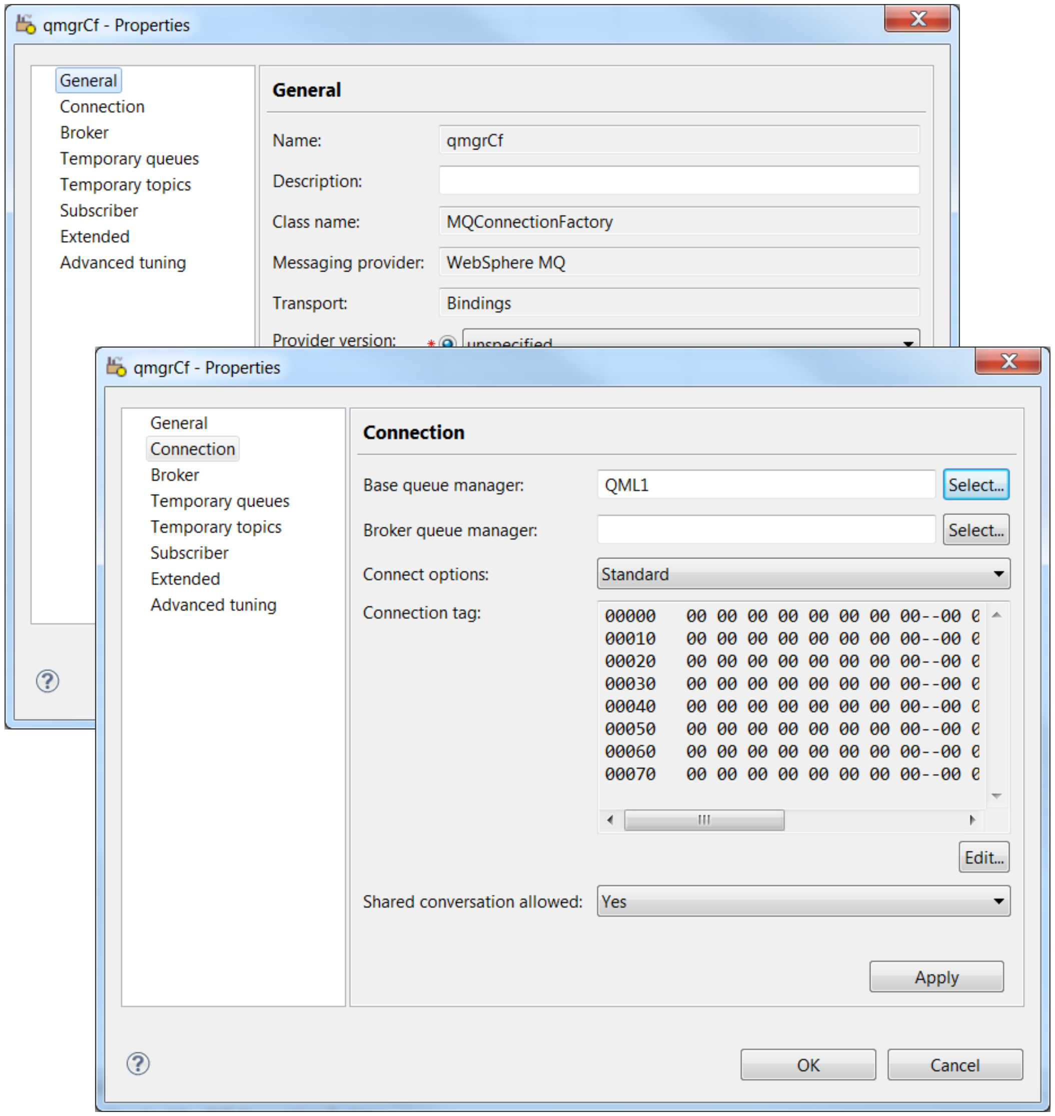
What is Java Naming and Directory Interface (JNDI)?

- *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

JMS Configuration Wizards in MQ Explorer



qmgrCf - Properties

- General
- Connection
- Broker
- Temporary queues
- Temporary topics
- Subscriber
- Extended
- Advanced tuning

General

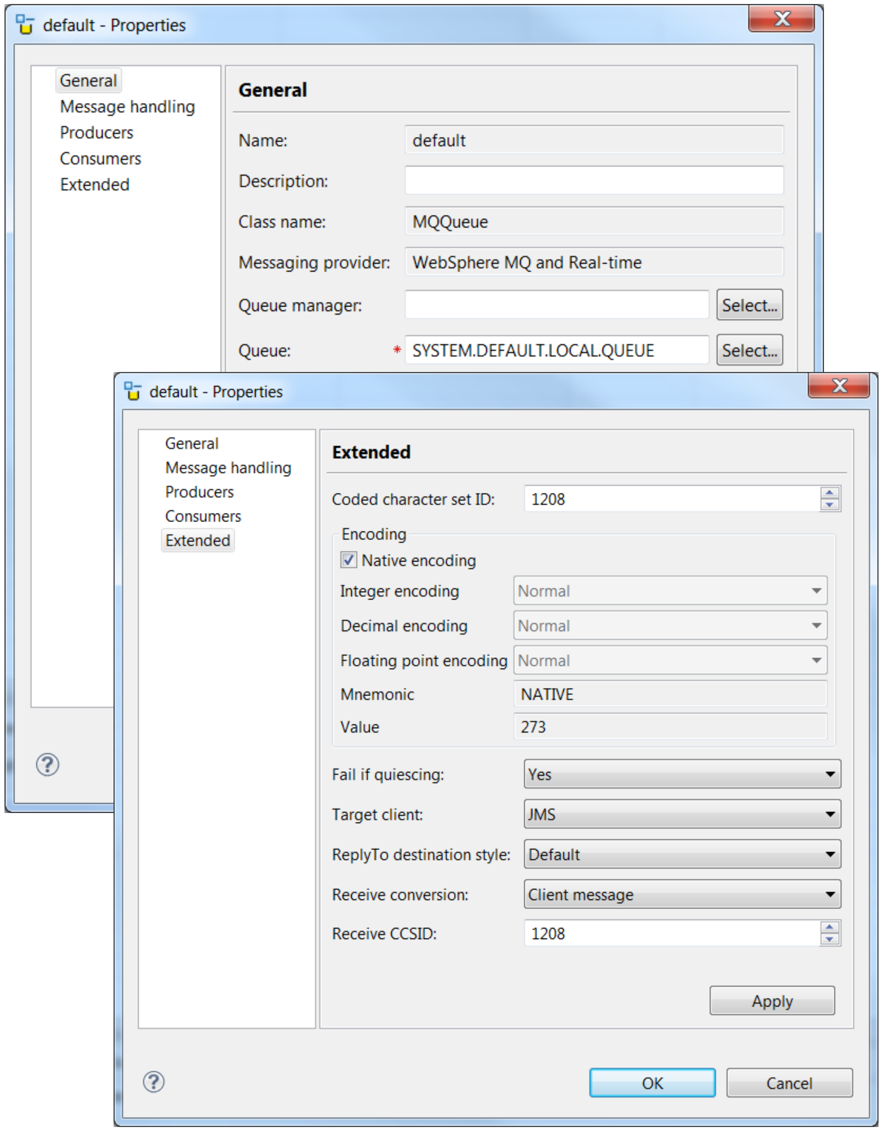
Name:	qmgrCf
Description:	
Class name:	MQConnectionFactory
Messaging provider:	WebSphere MQ
Transport:	Bindings
Provider version: *	unspecified

qmgrCf - Properties

Connection

Base queue manager:	QML1
Broker queue manager:	
Connect options:	Standard
Connection tag:	
Shared conversation allowed:	Yes

```
00000   00 00 00 00 00 00 00 00--00 @
00010   00 00 00 00 00 00 00 00--00 @
00020   00 00 00 00 00 00 00 00--00 @
00030   00 00 00 00 00 00 00 00--00 @
00040   00 00 00 00 00 00 00 00--00 @
00050   00 00 00 00 00 00 00 00--00 @
00060   00 00 00 00 00 00 00 00--00 @
00070   00 00 00 00 00 00 00 00--00 @
```



default - Properties

- General
- Message handling
- Producers
- Consumers
- Extended

General

Name:	default
Description:	
Class name:	MQQueue
Messaging provider:	WebSphere MQ and Real-time
Queue manager:	
Queue: *	SYSTEM.DEFAULT.LOCAL.QUEUE

default - Properties

Extended

Coded character set ID:	1208
Encoding	
☑ Native encoding	
Integer encoding	Normal
Decimal encoding	Normal
Floating point encoding	Normal
Mnemonic	NATIVE
Value	273
Fail if quiescing:	Yes
Target client:	JMS
ReplyTo destination style:	Default
Receive conversion:	Client message
Receive CCSID:	1208

Liberty server.xml Configuration Updates

- Add the connection factories queue manager and queues to the *server.xml* file

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

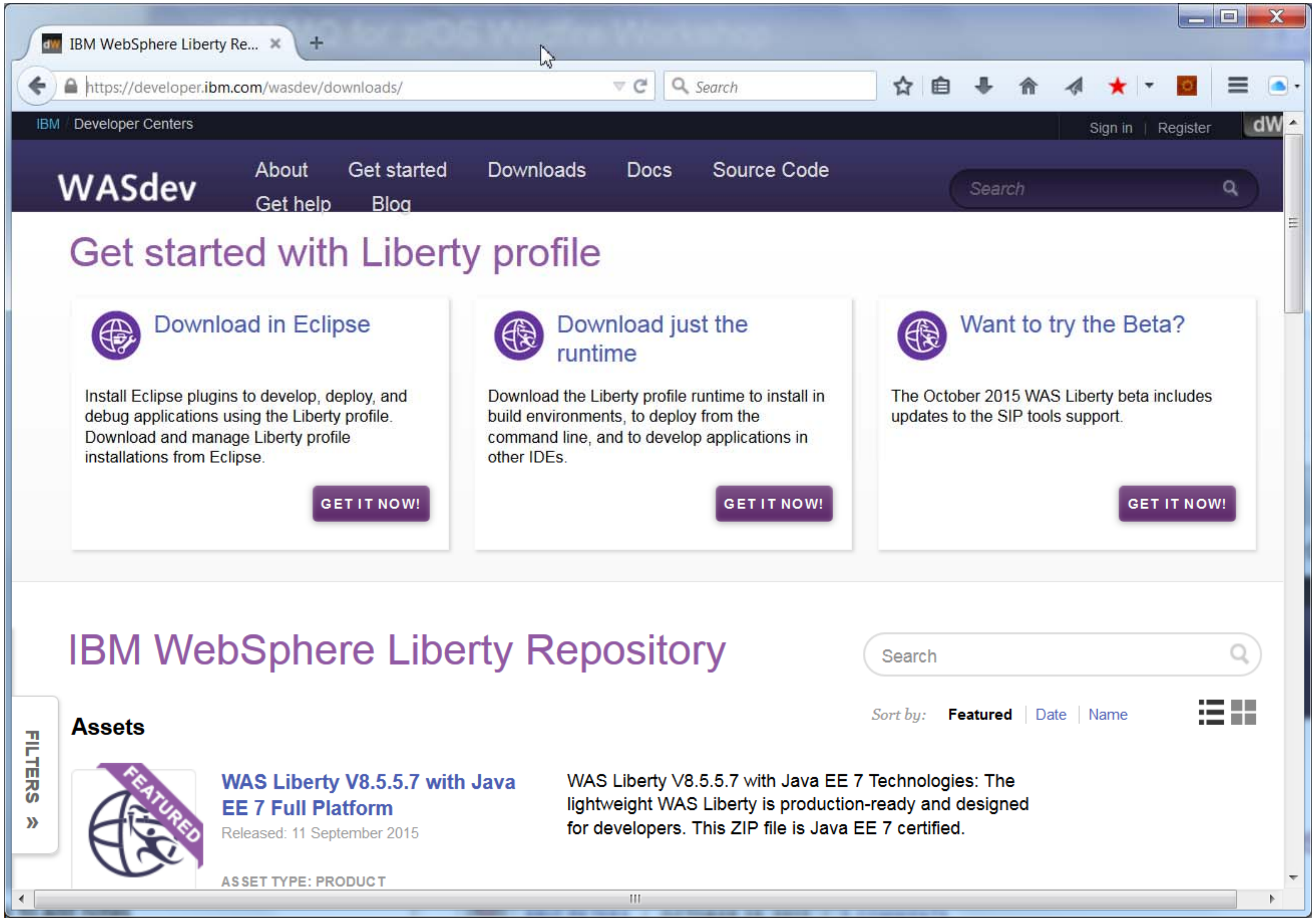
WebSphere Liberty Profile

Wildfire

Technical Hands-On Workshops

IBM zGrowth Team

Download a Liberty profile runtime



Get started with Liberty profile

Download in Eclipse

Install Eclipse plugins to develop, deploy, and debug applications using the Liberty profile. Download and manage Liberty profile installations from Eclipse.

GET IT NOW!

Download just the runtime

Download the Liberty profile runtime to install in build environments, to deploy from the command line, and to develop applications in other IDEs.

GET IT NOW!

Want to try the Beta?

The October 2015 WAS Liberty beta includes updates to the SIP tools support.

GET IT NOW!

IBM WebSphere Liberty Repository

Search

Assets

Sort by: **Featured** | Date | Name

FEATURED

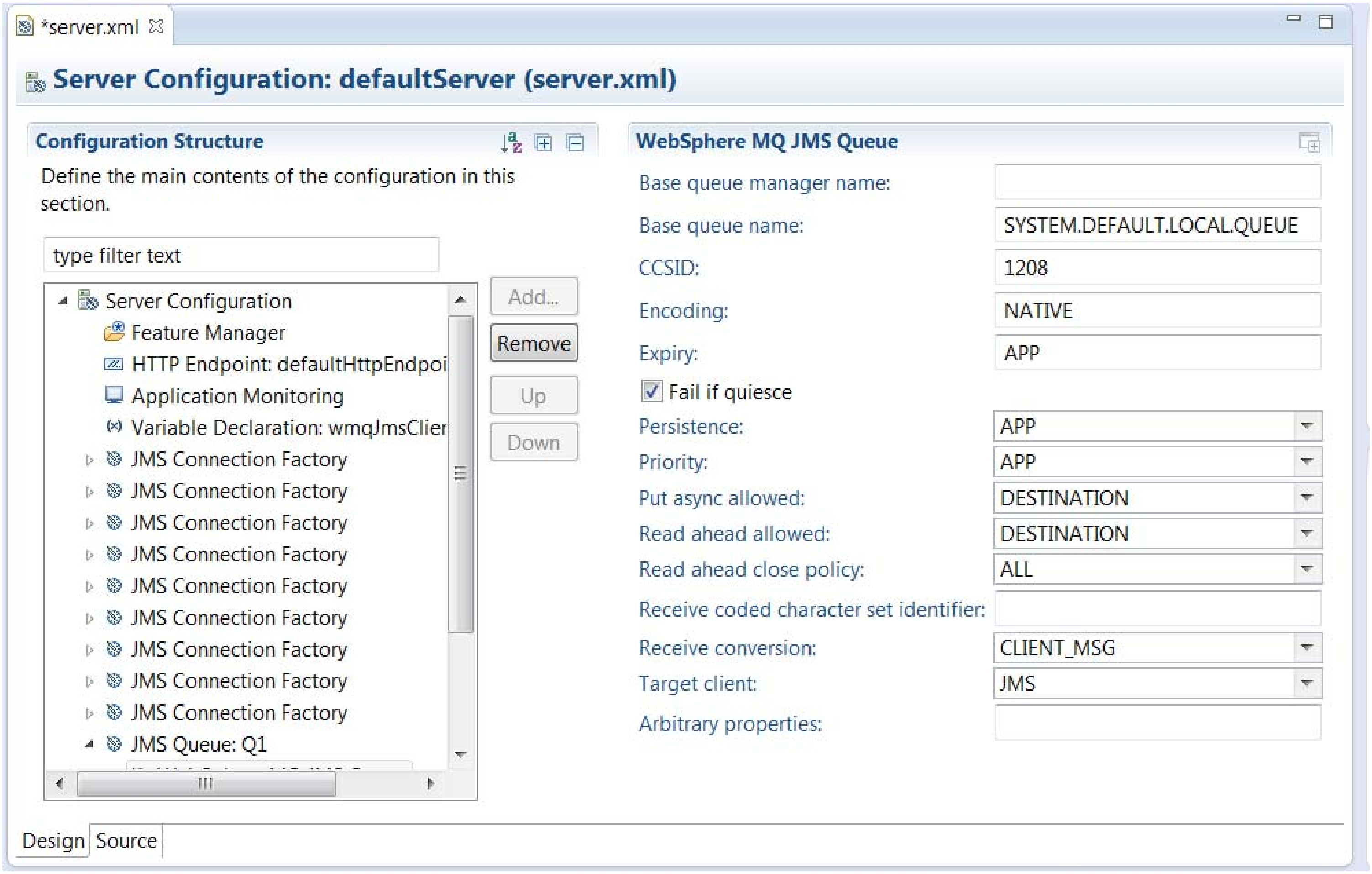
WAS Liberty V8.5.5.7 with Java EE 7 Full Platform

Released: 11 September 2015

ASSET TYPE: PRODUCT

WAS Liberty V8.5.5.7 with Java EE 7 Technologies: The lightweight WAS Liberty is production-ready and designed for developers. This ZIP file is Java EE 7 certified.

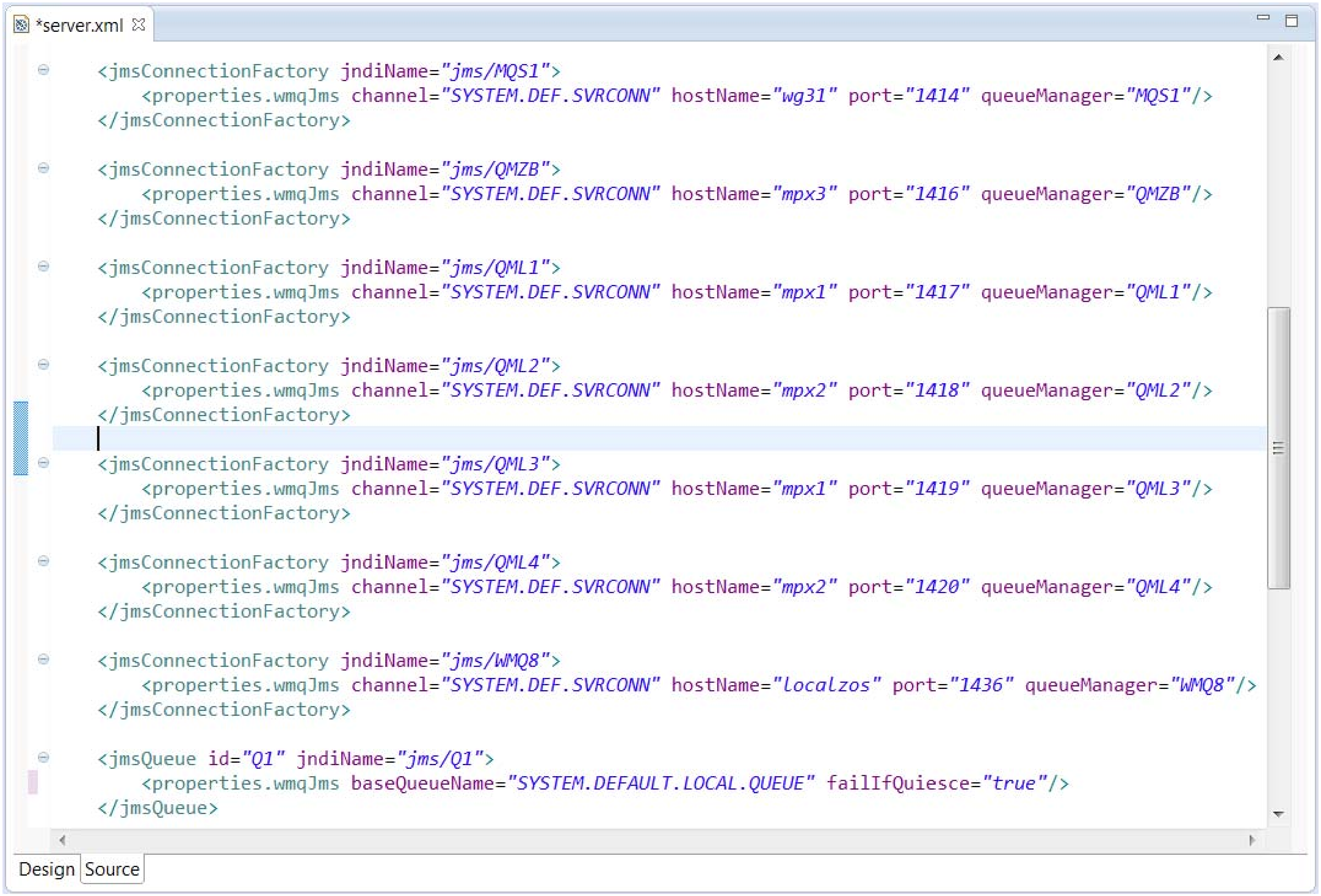
Liberty JMS Configuration Editor



WebSphere MQ JMS Queue

Base queue manager name:	
Base queue name:	SYSTEM.DEFAULT.LOCAL.QUEUE
CCSID:	1208
Encoding:	NATIVE
Expiry:	APP
✓ Fail if quiesce	
Persistence:	APP
Priority:	APP
Put async allowed:	DESTINATION
Read ahead allowed:	DESTINATION
Read ahead close policy:	ALL
Receive coded character set identifier:	
Receive conversion:	CLIENT_MSG
Target client:	JMS
Arbitrary properties:	

Liberty JMS Configuration Source



```
<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>
```

Integrating JMS in CICS and Liberty

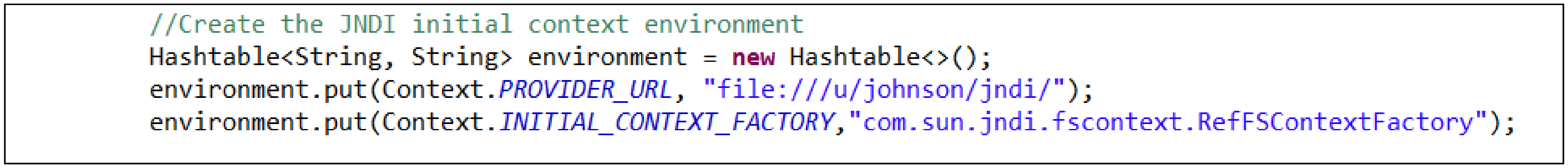
Wildfire
Technical Hands-On Workshops
IBM zGrowth Team

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 until CICS TS V5.4
  - 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

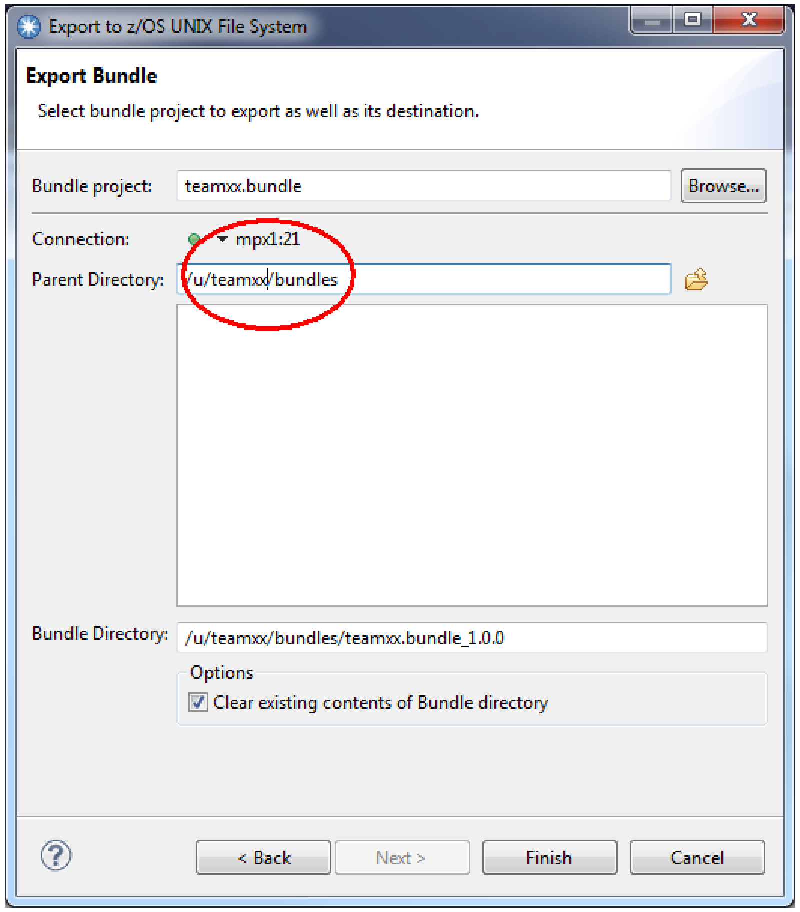
- Below is an example of the required updates and other relevant configuration variables.

```
WORK_DIR=/var/wlp/cics/MPX1CIC1 1

LIBPATH_SUFFIX=/shared/db2a10/jdbc/lib:/usr/lpp/mqm/V8R0M0/java/lib

OSGI_BUNDLES=/usr/lpp/mqm/V9R1M0/java/lib/OSGi/com.ibm.mq.osgi.allclientprereqs_9.1.0.2.jar,\ 2
          /usr/lpp/mqm/V9R1M0/java/lib/OSGi/com.ibm.mq.osgi.allclient_9.1.0.2.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



Select bundle project to export as well as its destination.

Bundle project: teamxx.bundle

Connection: mpx1:21

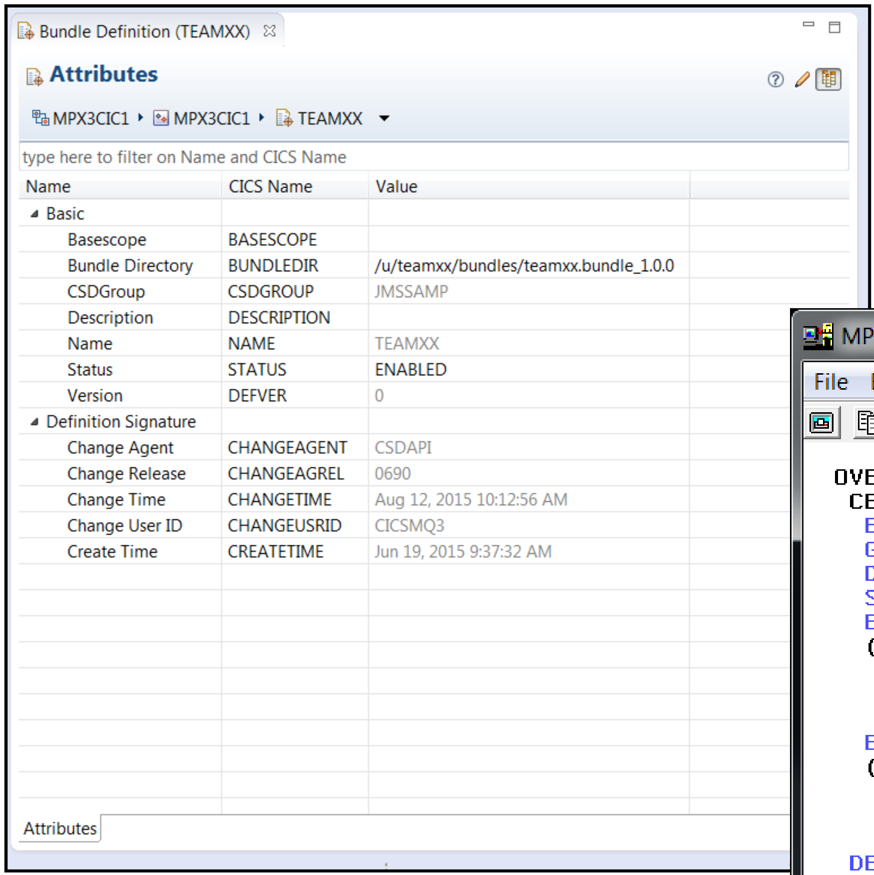
Parent Directory: /u/teamxx/bundles

Bundle Directory: /u/teamxx/bundles/teamxx.bundle_1.0.0

Options

[✓] Clear existing contents of Bundle directory

Required CICS Bundle Definition


Bundle Definition (TEAMXX)

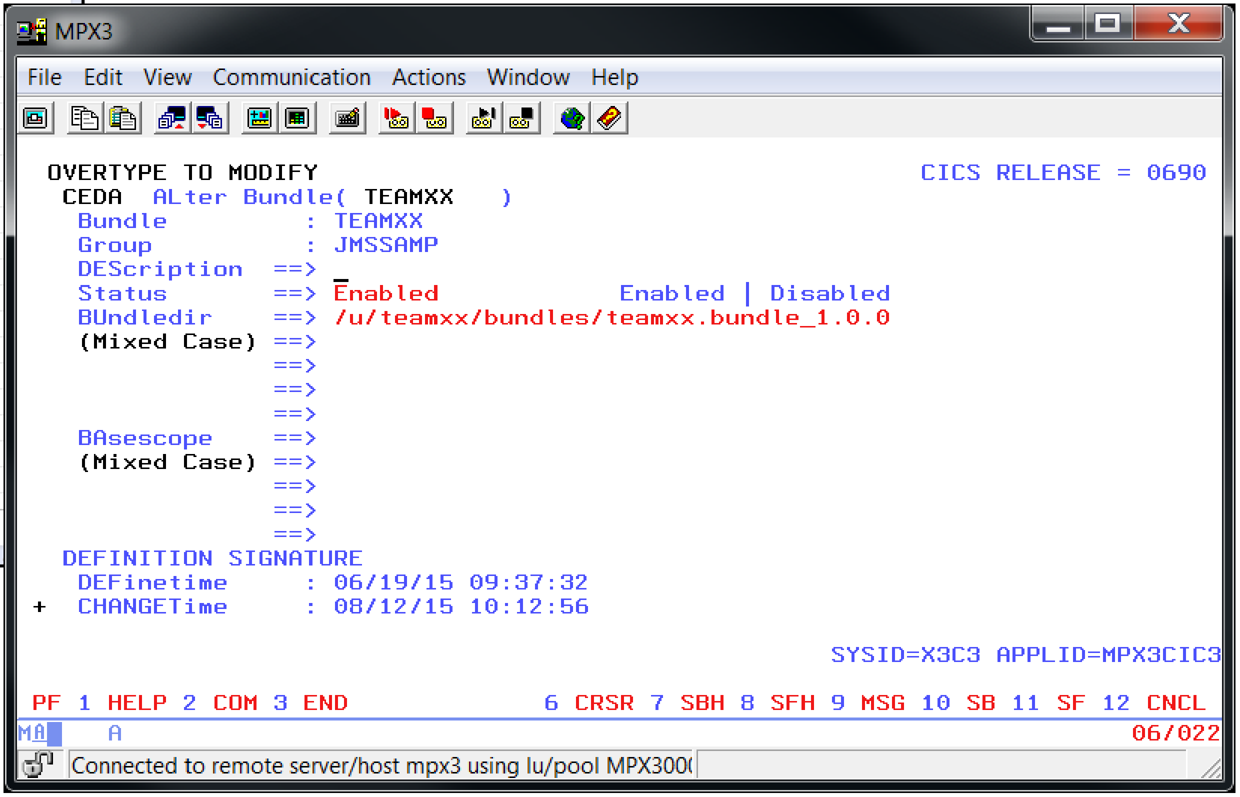
Attributes

MPX3CIC1 ▸ MPX3CIC1 ▸ 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	Aug 12, 2015 10:12:56 AM
Change User ID	CHANGEUSRID	CICSMQ3
Create Time	CREATETIME	Jun 19, 2015 9:37:32 AM

Attributes


MPX3

File Edit View Communication Actions Window Help

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0690
  CEDA  ALter 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      : 08/12/15 10:12:56

                                                  SYSID=X3C3 APPLID=MPX3CIC3
 PF 1 HELP 2 COM 3 END              6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
MA     A                                                                06/022
```

Connected to remote server/host mpx3 using lu/pool MPX300(

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();
}
```

Liberty server.xml Updates

- Liberty plugins for JMS can be download from URL

  https://developer.ibm.com/wasdev/downloads/#filter/sortby=relevance;q=jms

- Add the wmqJmsClient-1.1 and JNDI lookup features to the *server.xml* file

```
 <featureManager>
           <feature>wmqJmsClient-1.1</feature>
           <feature>jndi-1.0</feature>
</featureManager>
```

- Identify the location and name of the IBM MQ Resource Archival (RAR) file in the *server.xml* file

```
<variable name="wmqJmsClient.rar.location“value="/var/wlp/wmq/wmq.jmsra.rar"/>
```

MQ for z/OS Service Provider for z/OS Connect

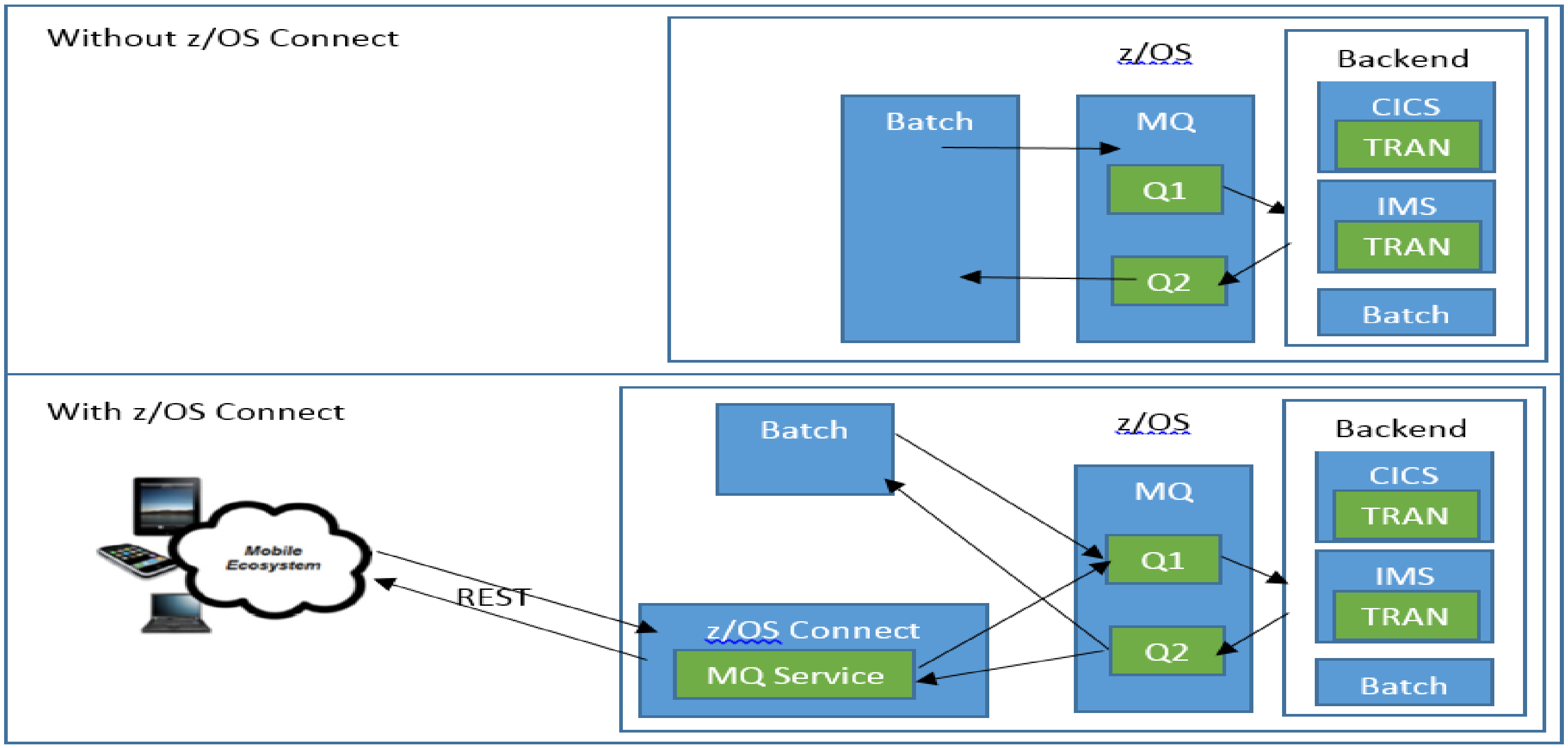
Wildfire

Technical Hands-On Workshops

IBM zGrowth Team

The MQ Service Provider

- Free of charge z/OS Connect service provider that allows existing services that are fronted by MQ to be accessed via a RESTful front end
  - Both z/OS Connect V1 and z/OS Connect EE V2 are supported
  - Same capabilities in both versions
- Clients need to have no knowledge of MQ



Service types

- Each URL in z/OS Connect maps to a service
- With the MQ Service Provider there are two different types of service
  - Two way services
  - One way services
- A two way service provides request/reply messaging:
  1. Client issues HTTP POST with some payload (JSON)
  2. MQ Service Provider sends payload (optional transformation) to one MQ queue
  3. Back end application processes payload and puts response on reply MQ queue
  4. MQ Service Provider gets response (optional transformation) and sends it to client as the body of the HTTP POST response
- A one way service exposes standard MQ verbs against a single destination
  - HTTP POST == MQPUT (queue and topic)
  - HTTP DELETE == MQGET (queue)
  - HTTP GET == MQGET (browse) (queue)

COBOL versus JSON Example

```
01 MINILOAN-COMMAREA.
          10 name pic X(20).
          10 creditScore pic 9(16)V99.
          10 yearlyIncome pic 9(16)v99.
          10 age pic 9(10).
          10 amount pic  9999999v99.
          10 approved pic X.
              88 BoolValue value 'T'.
          10 effectDate pic X(8).
          10 yearlyInterestRate pic S9(5).
          10 yearlyRepayment pic 9(18).
          10 messages-Num pic 9(9).
          10 messages pic X(60) occurs 1 to 99 times
                        depending on messages-Num.
```

```
"miniloan_commarea":{
              "type":"object",
              "properties":{
                 "name":{
                    "type":"string",
                    "maxLength":20
                 },
                 "creditScore":{
                    "type":"number",
                    "format":"decimal",
                    "multipleOf":0.01,
                    "maximum":9999999999999999.99,
                    "minimum":0
                 },
```

COBOL Source v JSON

“name”:”Mitch Johnson”, “creditScore”:100

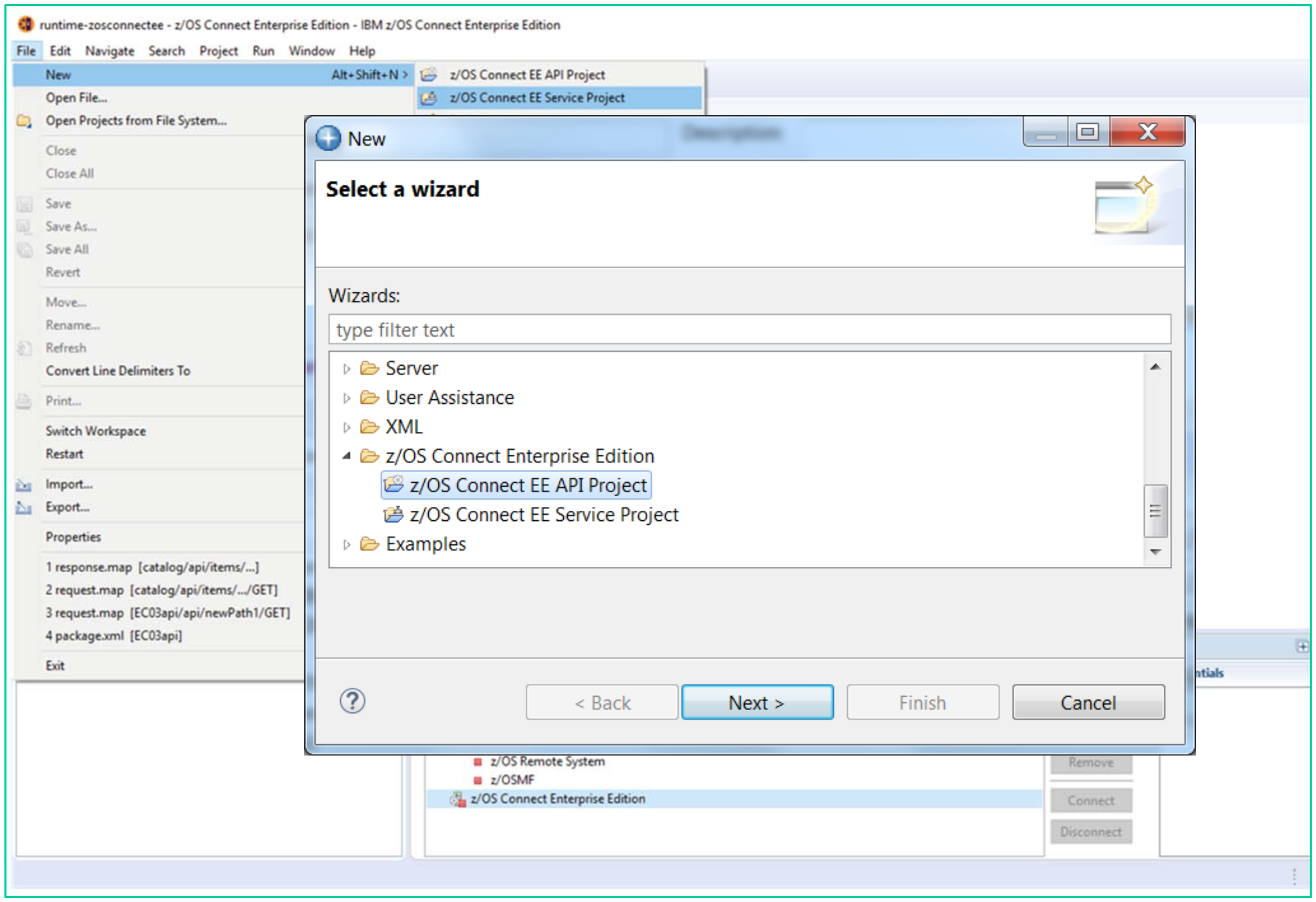
All data is sent as character strings and numeric precision and sign bit is removed as an issue

Generating z/OS Connect EE artifacts

```
//ASSIST   EXEC PGM=BPXBATCH
//STDPARM DD DSN=USER1.ZCEE.INPUT(MINILOAN),DISP=SHR
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD *
JAVA_HOME=/usr/lpp/java/J8.0_64
//
```

```
PGM /usr/lpp/IBM/zosconnect/v2r0/bin/baqls2js
PDSLIB=USER1.ZCEE.CNTL
REQMEM=MINILOAN
RESPMEM=MINILOAN
MAPPING-LEVEL=4.0
DATA-TRUNCATION=ENABLED
CHAR-VARYING=COLLAPSE
JSON-SCHEMA-REQUEST=/var/zosconnect/servers/server1/dataXform/json/Miniloan_request.json
JSON-SCHEMA-RESPONSE=/var/zosconnect/servers/server1/dataXform/json/Miniloan_response.json
LANG=COBOL
LOGFILE=/var/zosconnect/servers/server1/dataXform/Miniloan.log
WSBIND=/var/zosconnect/servers/server1/dataXform/bind/Miniloan.wsbind
SERVICE-ARCHIVE=/var/zosconnect/servers/server1/dataXform/sars/Miniloan.sar
SERVICE-NAME=MiniloanService
```

API toolkit – Creating Services for CICS and IMS

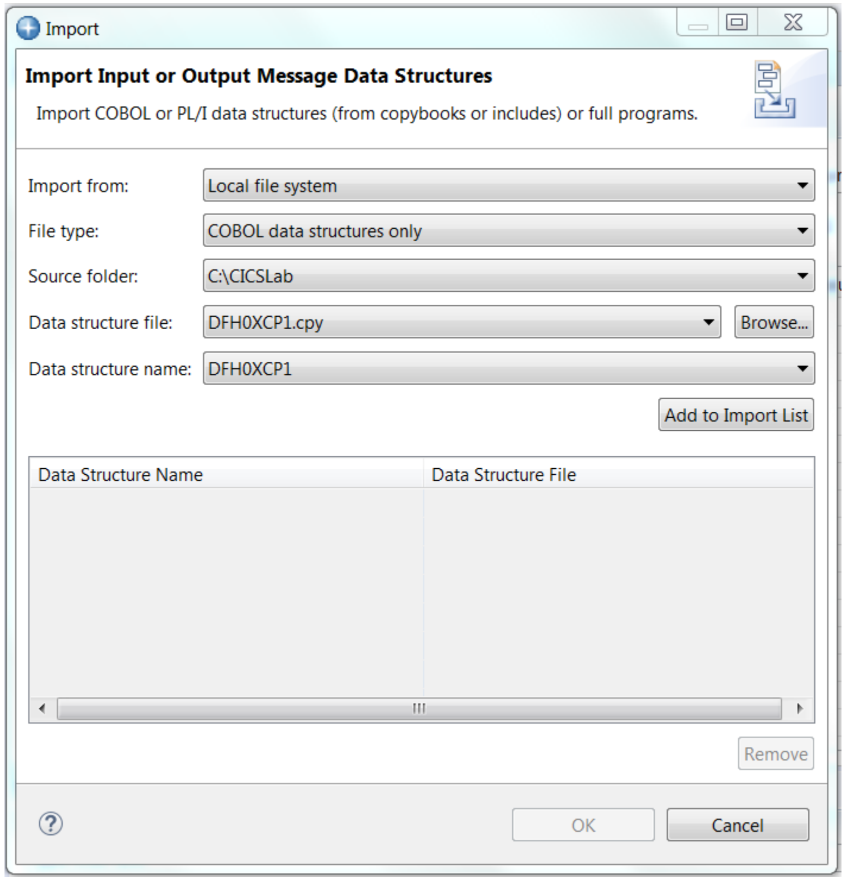

runtime-zosconnectee - z/OS Connect Enterprise Edition - IBM z/OS Connect Enterprise Edition
File Edit Navigate Search Project Run Window Help
New Alt+Shift+N >
z/OS Connect EE API Project
z/OS Connect EE Service Project
Open File...
Open Projects from File System...
Close
Close All
Save
Save As...
Save All
Revert
Move...
Rename...
Refresh
Convert Line Delimiters To
Print...
Switch Workspace
Restart
Import...
Export...
Properties
1 response.map [catalog/api/items/...]
2 request.map [catalog/api/items/.../GET]
3 request.map [EC03api/api/newPath1/GET]
4 package.xml [EC03api]
Exit
New
Select a wizard
Wizards:
type filter text
Server
User Assistance
XML
z/OS Connect Enterprise Edition
z/OS Connect EE API Project
z/OS Connect EE Service Project
Examples
< Back
Next >
Finish
Cancel
z/OS Remote System
z/OSMF
z/OS Connect Enterprise Edition
Remove
Connect
Disconnect

Use the **API toolkit** to create services through Eclipse-based tooling.

Services are described as **Projects**, so They can be easily managed in source control.

API toolkit – Creating Services for CICS and IMS

- Creating a service project



Import

Import Input or Output Message Data Structures

Import COBOL or PL/I data structures (from copybooks or includes) or full programs.

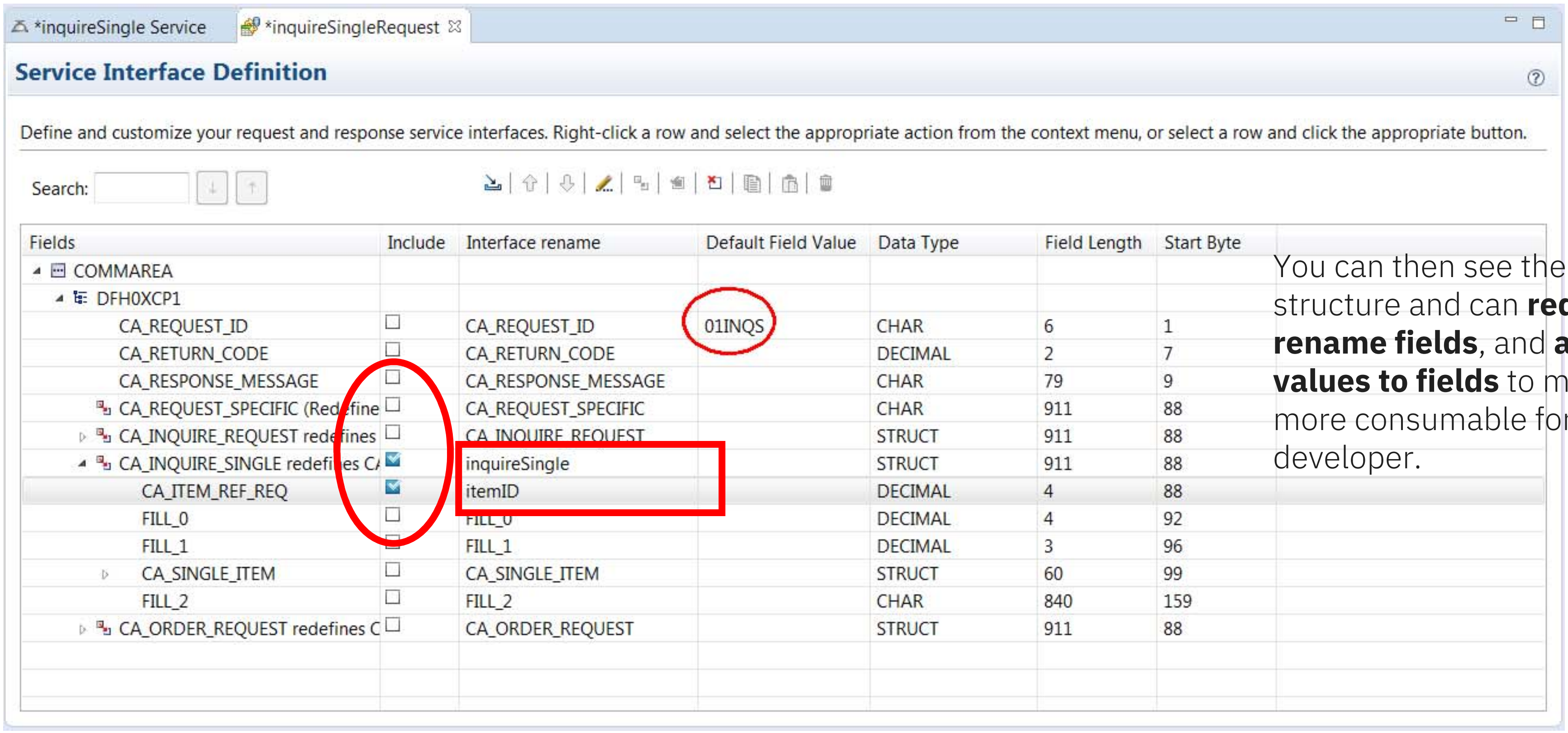
Import from:	Local file system
File type:	COBOL data structures only
Source folder:	C:\CICSLab
Data structure file:	DFH0XCP1.cpy
Data structure name:	DFH0XCP1

You start by importing data structures into the service interface from the local file system or the workspace.

The service interface supports complex data structures, including OCCURS DEPENDING ON and REDEFINES clauses.

API toolkit – Creating Services for CICS and IMS

- Creating a service interface definition



*inquireSingle Service | *inquireSingleRequest

Service Interface Definition

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

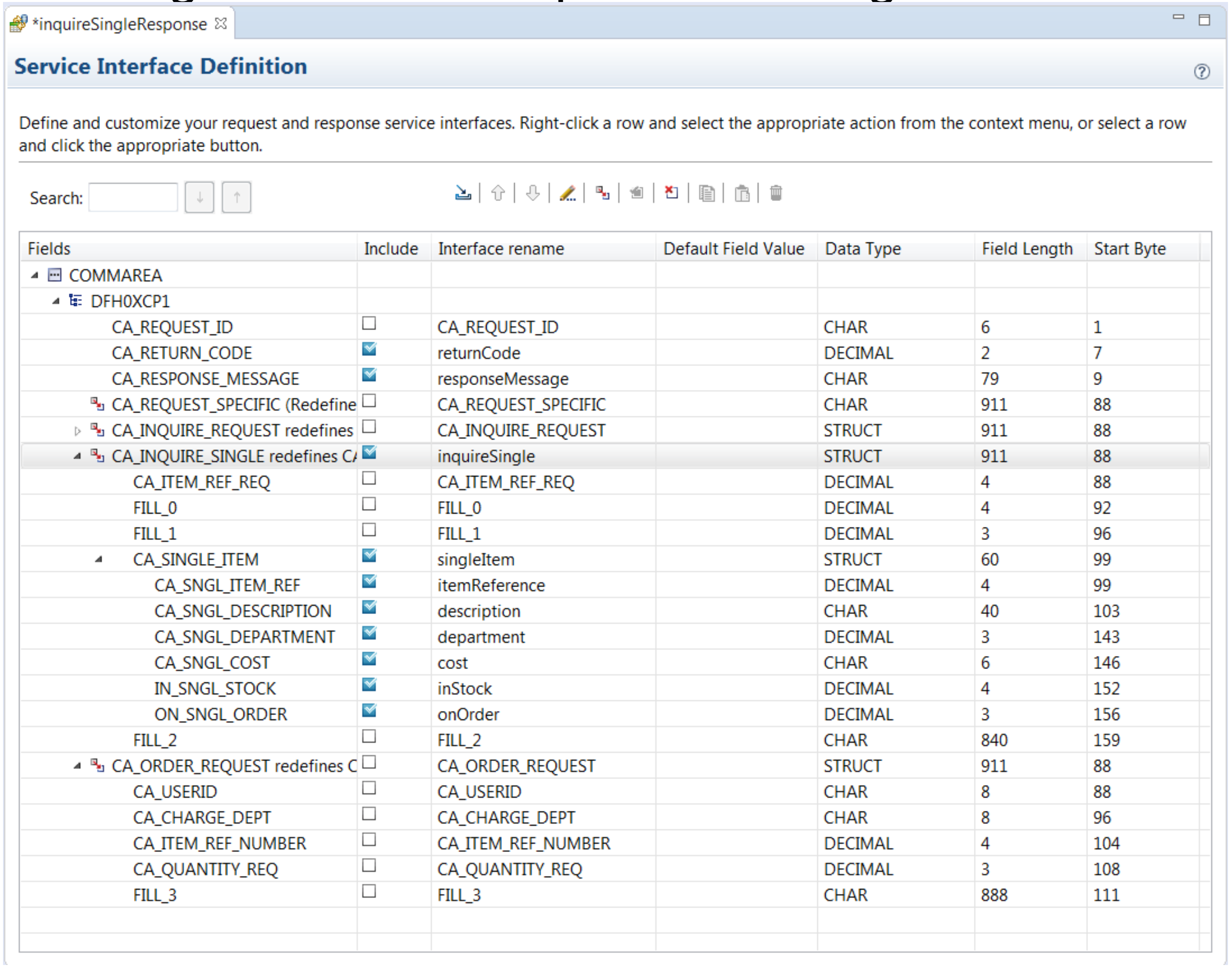
Search:

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	☐	CA_REQUEST_ID	01INQS	CHAR	6	1
CA_RETURN_CODE	☐	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	☐	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefine	☐	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines	☐	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines CA	☑	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	☑	itemID		DECIMAL	4	88
FILL_0	☐	FILL_0		DECIMAL	4	92
FILL_1	☐	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	☐	CA_SINGLE_ITEM		STRUCT	60	99
FILL_2	☐	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefines C	☐	CA_ORDER_REQUEST		STRUCT	911	88

You can then see the imported data structure and can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.

API toolkit – Creating Services for CICS and IMS

- Creating a service – response message



*inquireSingleResponse

Service Interface Definition

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

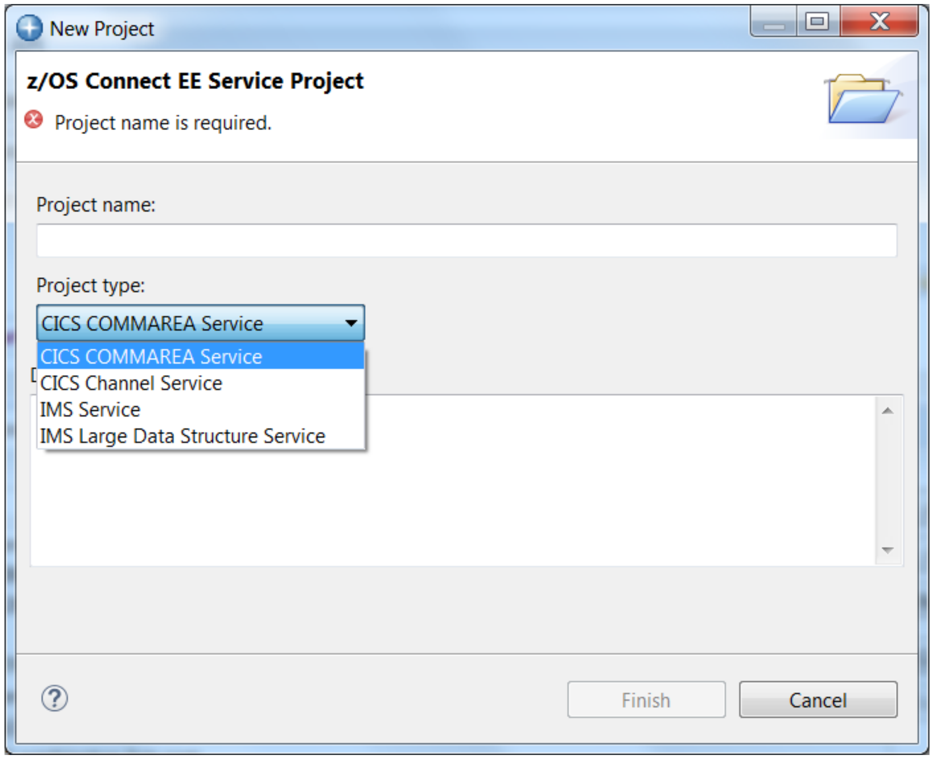
Search:

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	☐	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	✓	returnCode		DECIMAL	2	7
CA_RESPONSE_MESSAGE	✓	responseMessage		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefine	☐	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines	☐	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines C/	✓	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	☐	CA_ITEM_REF_REQ		DECIMAL	4	88
FILL_0	☐	FILL_0		DECIMAL	4	92
FILL_1	☐	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	✓	singleItem		STRUCT	60	99
CA_SNGL_ITEM_REF	✓	itemReference		DECIMAL	4	99
CA_SNGL_DESCRIPTION	✓	description		CHAR	40	103
CA_SNGL_DEPARTMENT	✓	department		DECIMAL	3	143
CA_SNGL_COST	✓	cost		CHAR	6	146
IN_SNGL_STOCK	✓	inStock		DECIMAL	4	152
ON_SNGL_ORDER	✓	onOrder		DECIMAL	3	156
FILL_2	☐	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefines C	☐	CA_ORDER_REQUEST		STRUCT	911	88
CA_USERID	☐	CA_USERID		CHAR	8	88
CA_CHARGE_DEPT	☐	CA_CHARGE_DEPT		CHAR	8	96
CA_ITEM_REF_NUMBER	☐	CA_ITEM_REF_NUMBER		DECIMAL	4	104
CA_QUANTITY_REQ	☐	CA_QUANTITY_REQ		DECIMAL	3	108
FILL_3	☐	FILL_3		CHAR	888	111

You can then see the imported data structure and can **redact fields** and **rename fields**

API toolkit – Creating Services for CICS and IMS

- Service creation – a common interface



New Project

z/OS Connect EE Service Project

Project name is required.

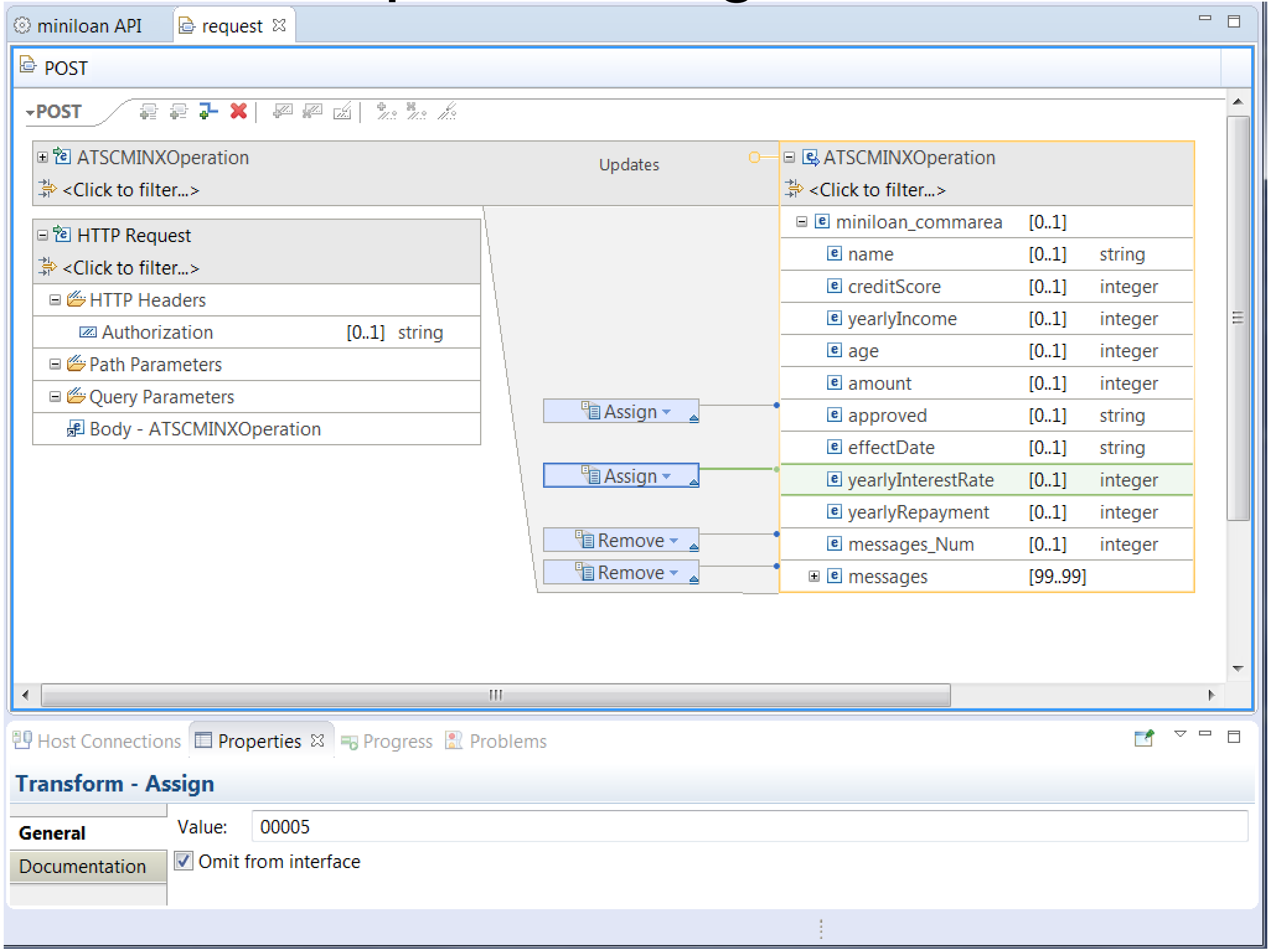
Project name:

Project type:

CICS COMMAREA Service

A common interface for service creation, agnostic of back end subsystem.

API Editor Eclipse Tooling



Authorization	[0..1]	string

ATSCMINXOperation

miniloan_commarea	[0..1]	
name	[0..1]	string
creditScore	[0..1]	integer
yearlyIncome	[0..1]	integer
age	[0..1]	integer
amount	[0..1]	integer
approved	[0..1]	string
effectDate	[0..1]	string
yearlyInterestRate	[0..1]	integer
yearlyRepayment	[0..1]	integer
messages_Num	[0..1]	integer
messages	[99..99]	

Transform - Assign

Value: 00005

[✓] Omit from interface

Two way Service Elements

server.xml

```
<zosconnect_zosConnectService id="miniloan"
        dataXformRef="xformJSON2Byte"
        serviceName="Miniloan"
        serviceDescription="MQ Reply/Response Service"
        serviceRef="Miniloan" />

<mqzosconnect_mqzOSConnectService id="Miniloan"
       connectionFactory="jms/qmgrCf"
       waitInterval="30000"
       destination="jms/request"
       replyDestination="jms/response"/>
```

Provided by z/OS Connect

Provided by MQ Service Provider

- HTTP POST to **https://<hostname>:<port>/miniloan**
- All MQ related information is held in *mqzOSConnectService element*
  - Sensible defaults
  - Overridable via HTTP headers, e.g. *ibm-mq-gmo-waitInterval*
  - Builds on the MQ messaging provider in Liberty. Uses JMS

Queue Manager and Queue Elements

```
<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
      connectionManagerRef="ConMgr1">
      <properties.wmqJMS transportType="BINDINGS"
               queueManager="QMZ1" />
 </jmsConnectionFactory>

<jmsQueue id="request" jndiName="jms/request">
     <properties.wmqJms
         baseQueueName="CICS.TRIGGER.REQUEST"
         targetClient="MQ"
         CCSID="37"/>
 </jmsQueue>

 <jmsQueue id="response" jndiName="jms/response">
     <properties.wmqJms
         baseQueueName="CICS.TRIGGER.RESPONSE"
         targetClient="MQ"
         CCSID="37"/>
```

Questions???