Message Queues 1.1

Summary: This document provides basic information about the extension providing **Message Queues** support for Java.

Extension ID

com.castsoftware.mqe

What's new?

Please see Message Queues 1.1 - Release Notes for more information.

Description

This extension should be installed when analyzing projects containing Message Queue applications, and you want to view a transaction consisting of queue calls and queue receive objects with their corresponding links. This version supports Message Queues for:

Java	Plain Java	✓
	Spring	✓
	JMS	✓
Mainframe	Supported via the Mainframe analyzer. See Support for IBM MQSeries	✓

Supported Message Queue versions

The following table displays the supported versions matrix:

Message Queue	Version	Support
ActiveMQ	5.15.3	• OpenWire + JMS • Spring + JMS with XML based configuration • JMS with SpringBoot

IBM MQ	6.0.0, 8.0.0	• Spring + JMS with XML and Annotation based configuration • SpringBoot (when queue is autowired in different file) • Plain Java
RabbitMQ	3.6.9	• AMQP + SLF4J • Spring AMQP + Spring Rabbit with XML based configuration • Spring AMQP with SpringBoot

CAST AIP Compatibility

This extension is compatible with:

CAST AIP release	Supported
8.3.x	✅
8.2.x	✅
8.1.x	✅
8.0.x	✅
7.3.4 and all higher 7.3.x releases	✅

Supported DBMS servers

This extension is compatible with the following DBMS servers:

DBMS	Supported
CSS	✅
Oracle	✅
Microsoft SQL Server	❌

Prerequisites

✅	An installation of any compatible release of CAST AIP (see table above)

Dependencies with other extensions

Some CAST extensions require the presence of other CAST extensions in order to function correctly. The **Message Queue** extension requires that the following other CAST extensions are also installed:

- Web Services Linker
- **CAST AIP Internal extension** (internal technical extension)

ℹ Note that when using the **CAST Extension Downloader** to download the extension and the **Manage Extensions** interface in **CAST Server Manager** to install the extension, any dependent extensions are **automatically** downloaded and installed for you. You do not need to do anything.

Download and installation instructions

Please see:

- Download an extension
- Install an extension

The latest release status of this extension can be seen when downloading it from the CAST Extend server.

Packaging, delivering and analyzing your source code

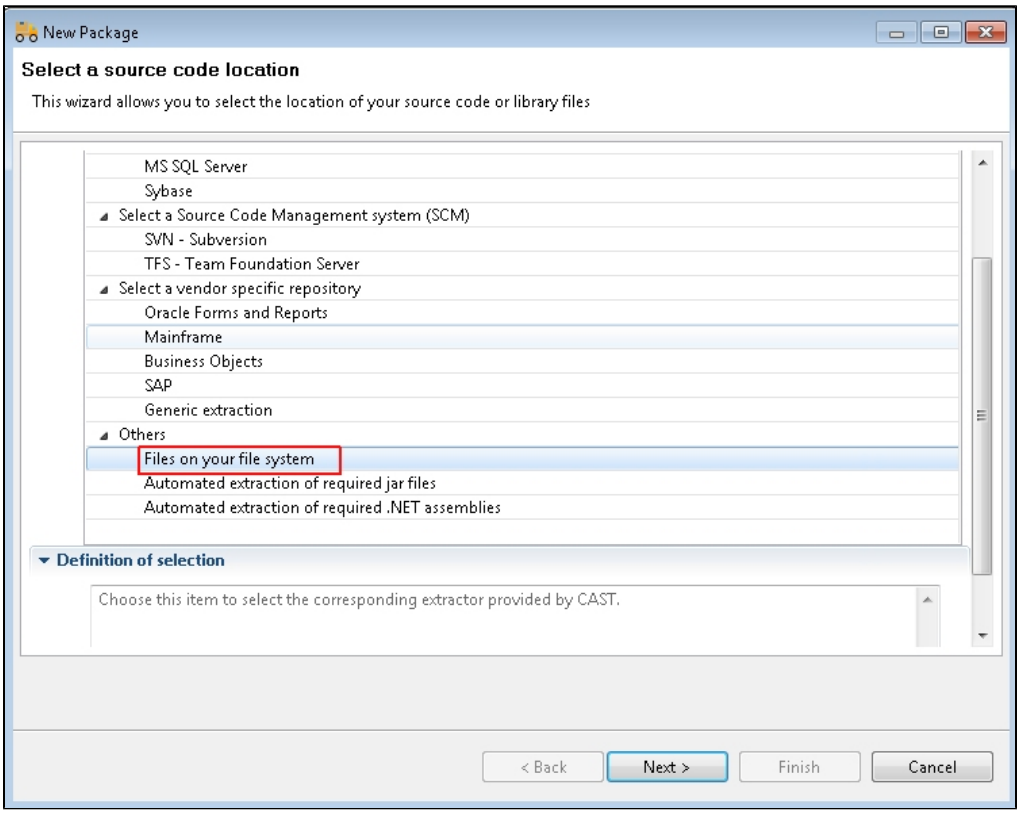
Once the extension is installed, no further configuration changes are required before you can package your source code and run an analysis. The process of packaging, delivering and analyzing your source code is as follows:

Packaging and delivery

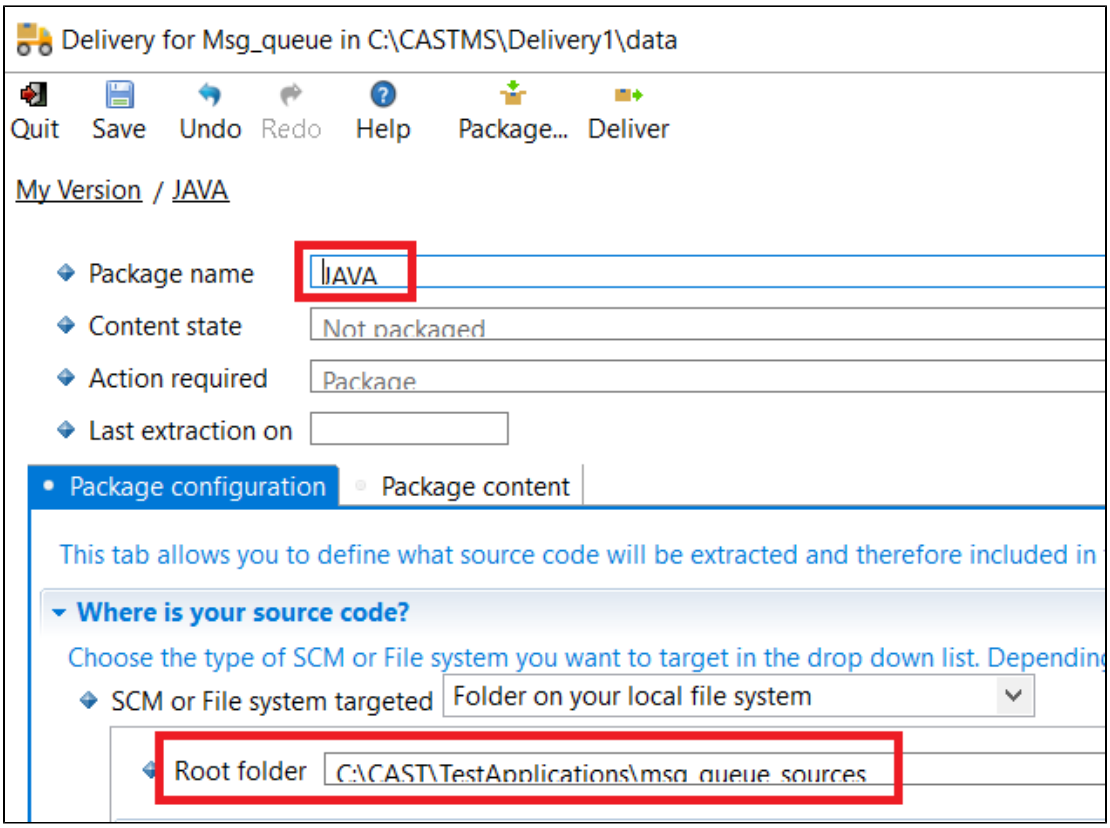
Note that the **Message Queues** extension does not contain any CAST Delivery Manager Tool **discoverers or extractors**, therefore, no "Message Queue" projects will be detected by the DMT. You therefore need to manually create an Analysis Unit in the CAST Management Studio - this is explained below.

Using the CAST Delivery Manager Tool:

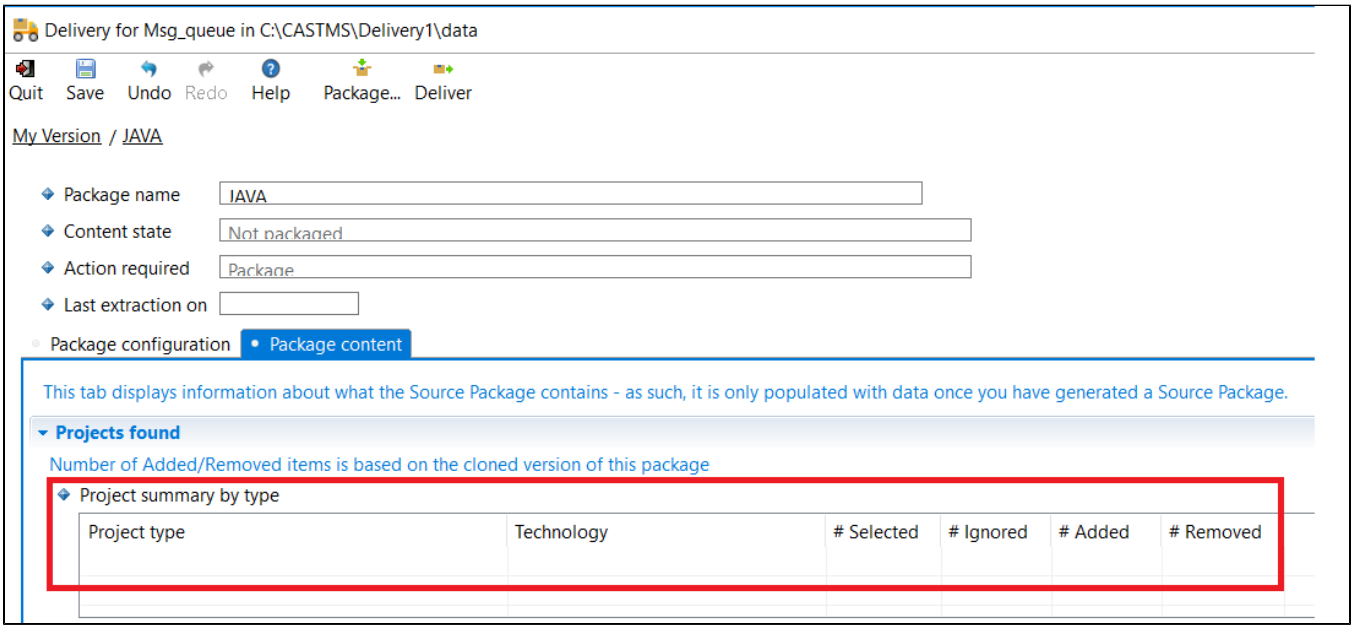
- Create a new **Version**
- Create a new **Package** for your Message Queue source code using the **Files on your file system** option:



- Define a **name** for the package and the **root folder** of your Application source code:



- Run the **Package action**: the CAST Delivery Manager Tool will **not** find any "projects" related to the Message Queue application source code - this is the **expected behavior**. However, if your Java related source code is part of a larger application, then other projects may be found during the package action.

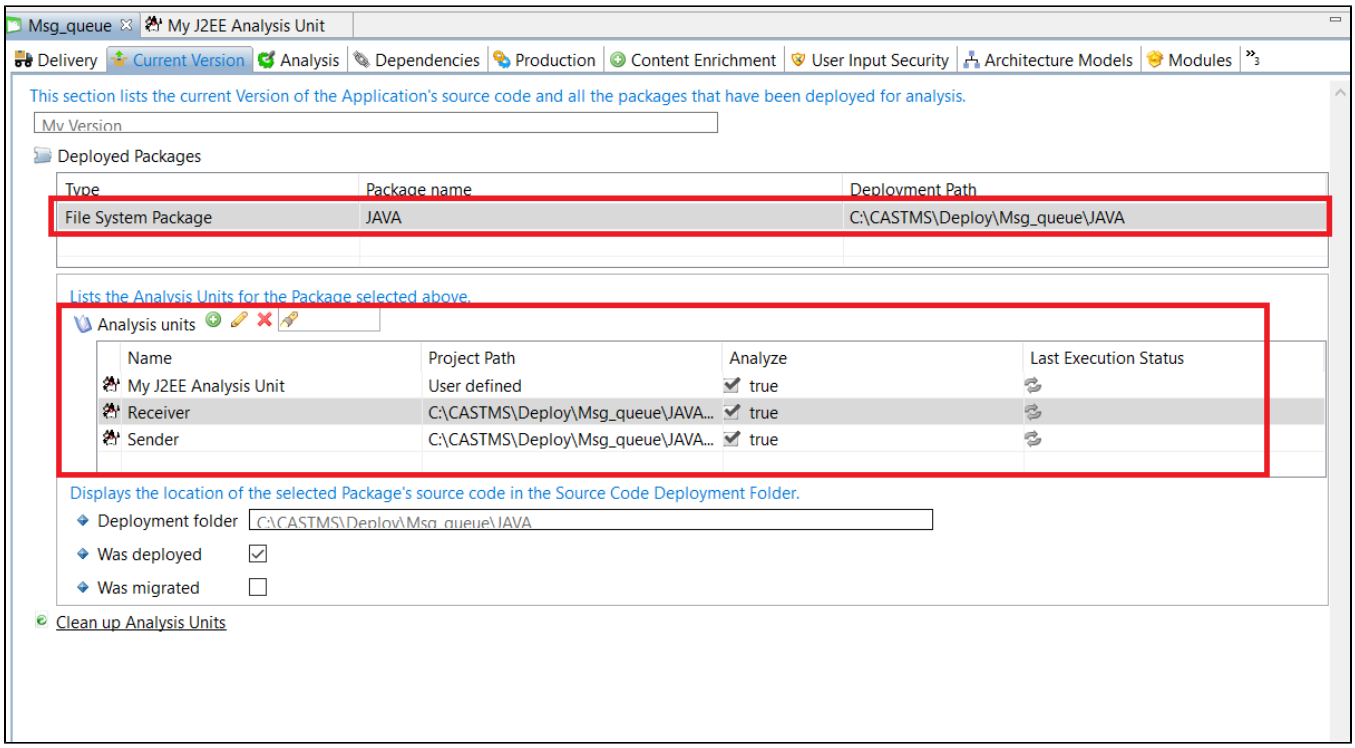


- Deliver the **Version**

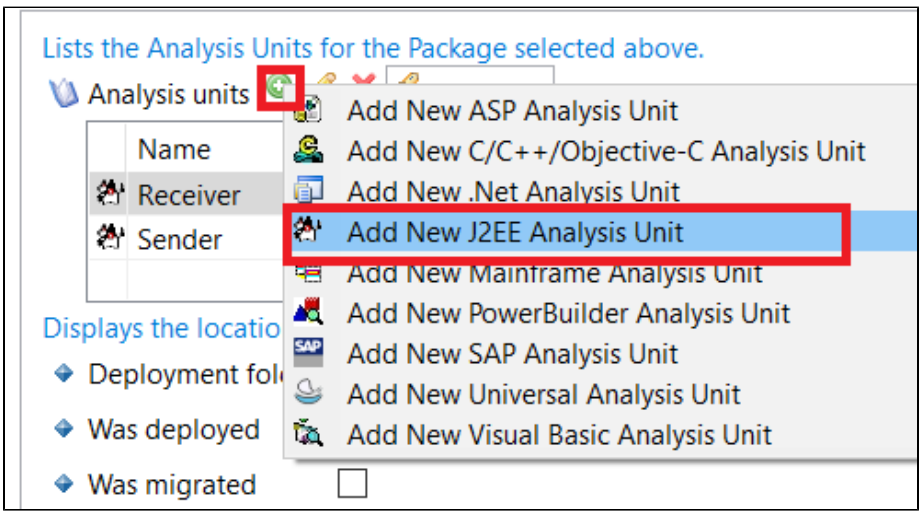
Analyzing

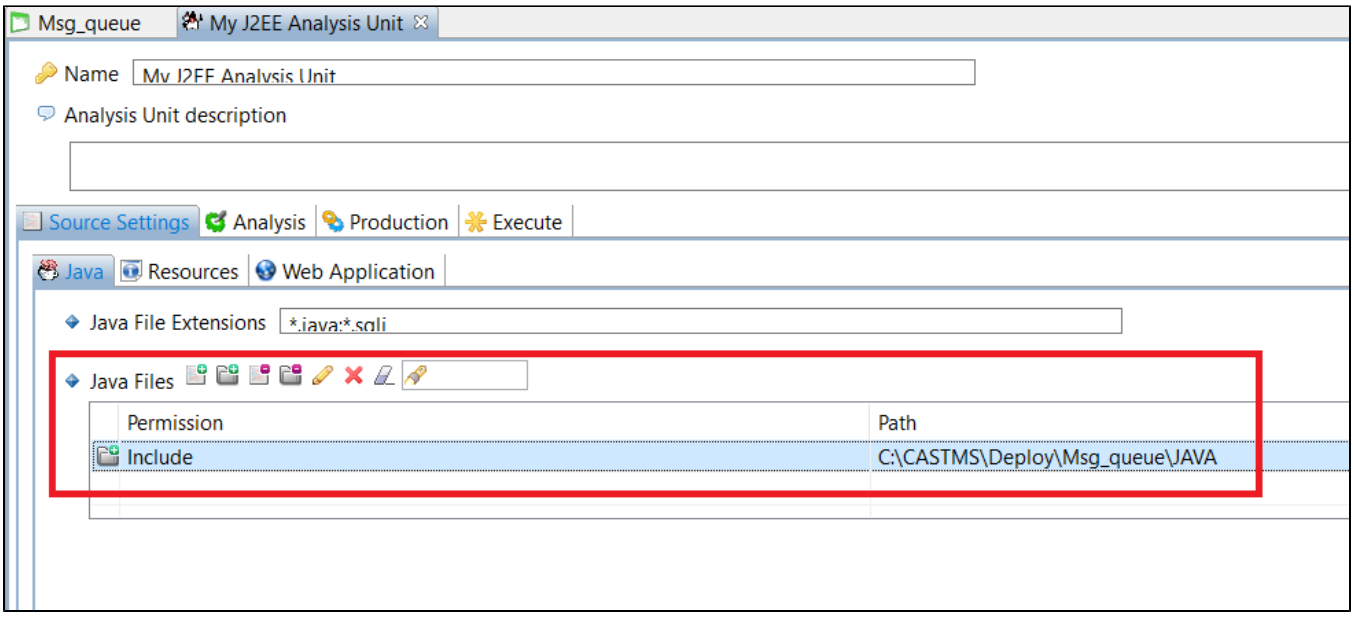
Using the CAST Management Studio:

- Accept and deploy the **Version** in the CAST Management Studio. No **Analysis Units** will be created automatically relating to the Java source code - this is the **expected behavior**. However, if your Message Queue related source code is part of a larger application, then other Analysis Units may be created automatically:



- In the **Current Version tab**, add a new Analysis Unit specifically for your Java source code containing Message Queues, selecting the **Add new J2EE Analysis Unit** option:

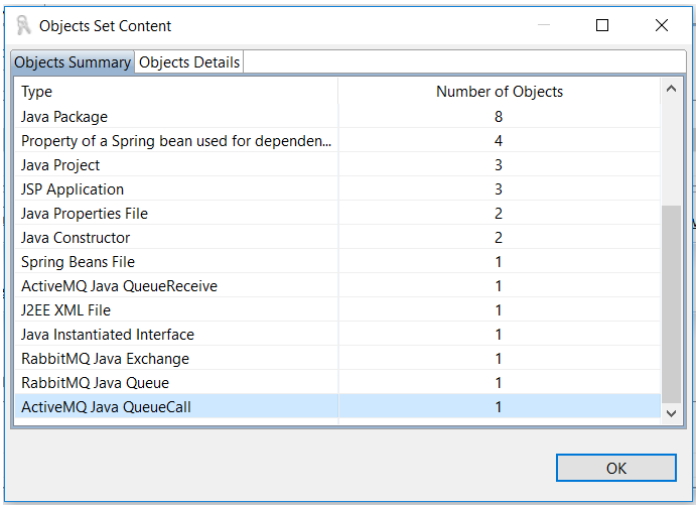


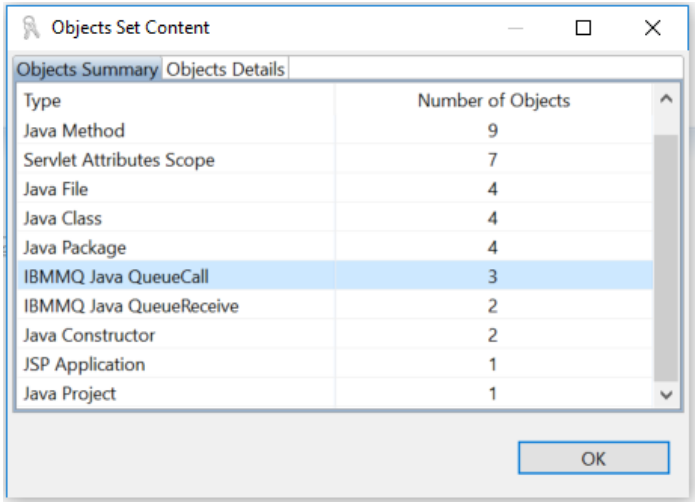


- Run a **test analysis** on the Analysis Unit before you generate a **new snapshot**.

What results can you expect?

Once the analysis/snapshot generation has completed, you can view the results in the normal manner:

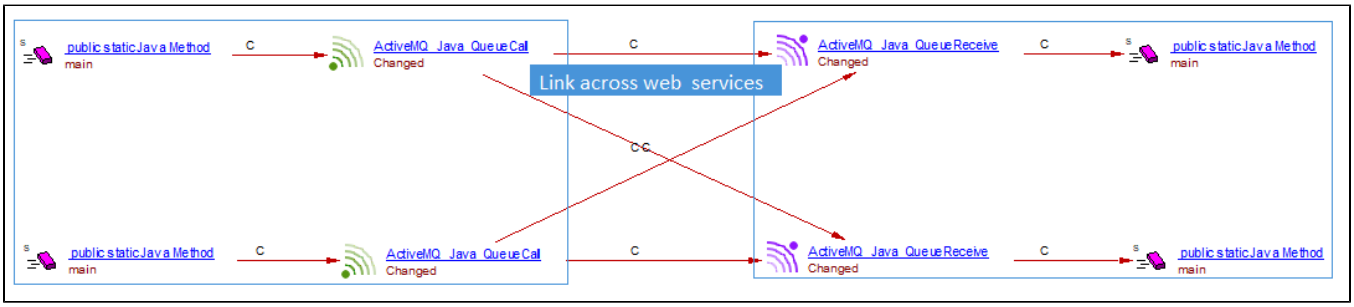




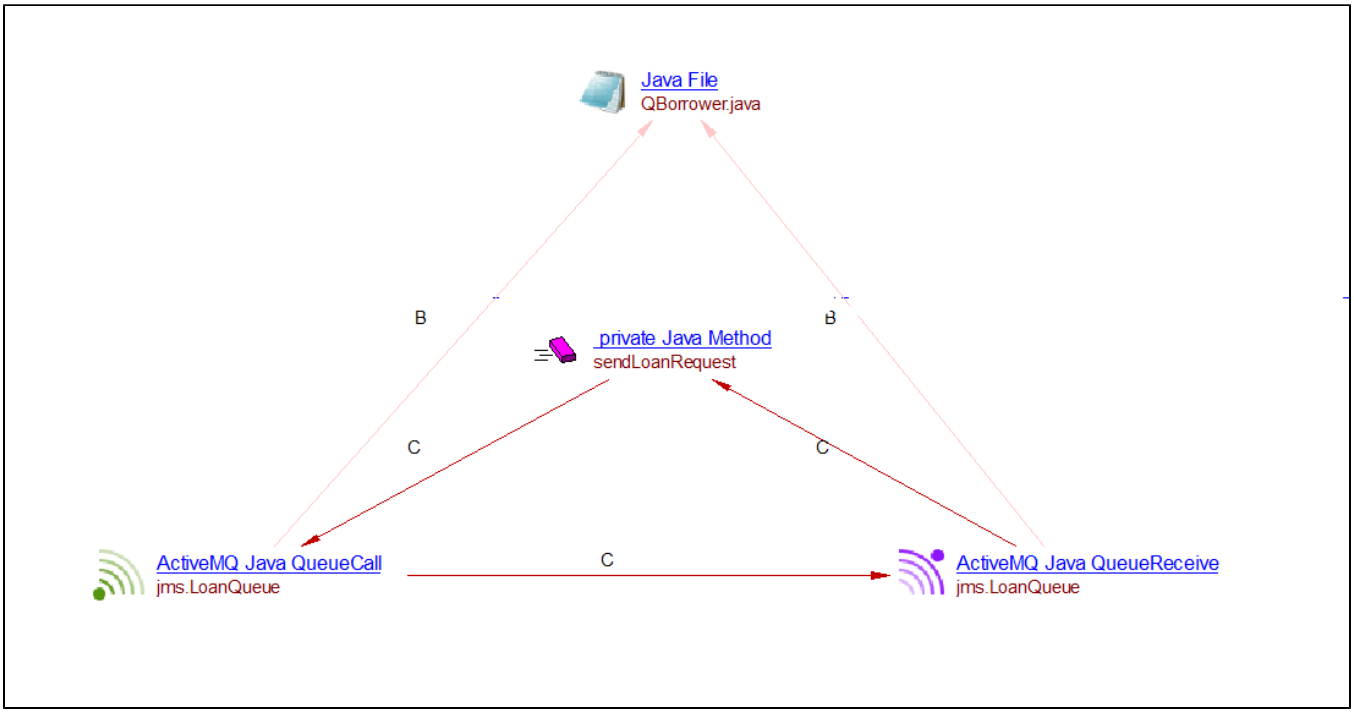
Below are the transactions obtained after analysis:

ActiveMQ

Many-to-Many : ActiveMQ

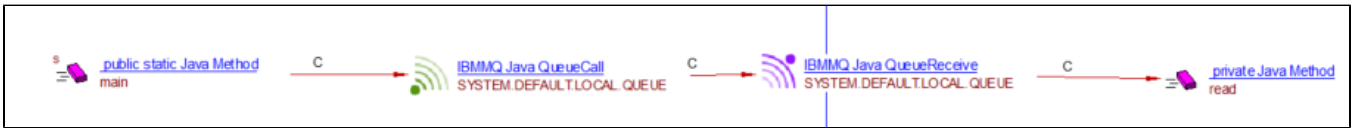


Producer/Consumer in same file: ActiveMQ

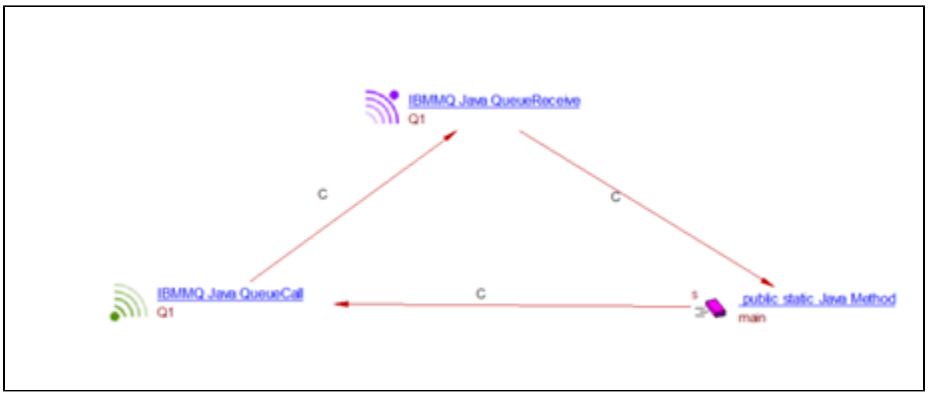


IBM MQ

Producer-to-Consumer : IBM MQ

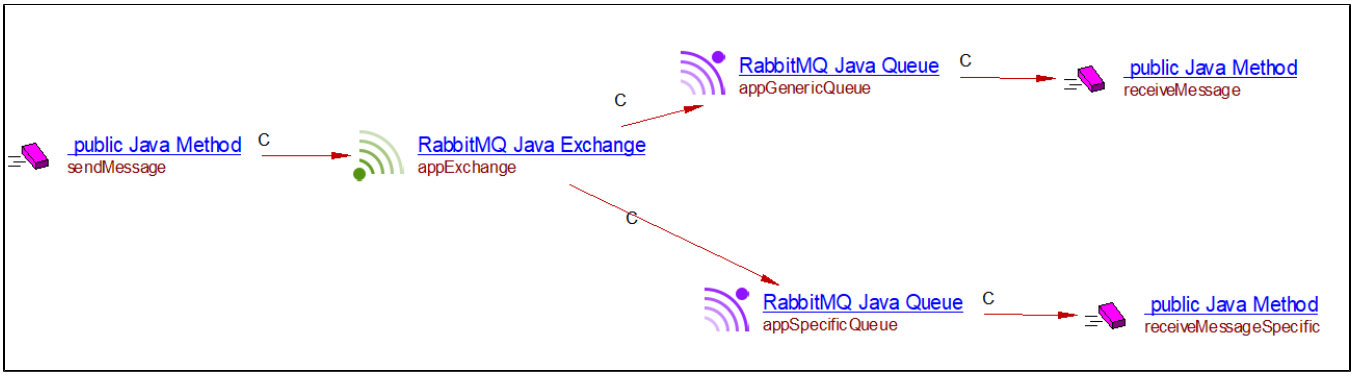


Producer/Consumer in same file : IBM MQ

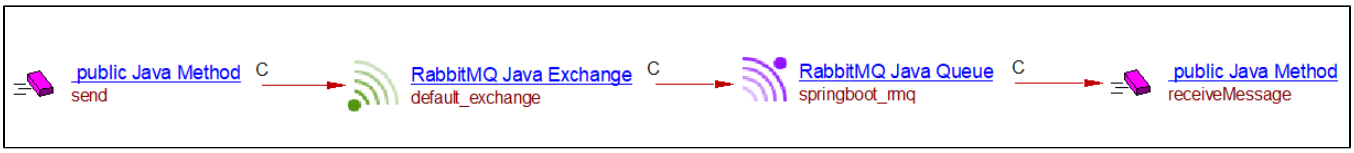


RabbitMQ

One to Many: RabbitMQ Topic Exchange bound to two Queues



RabbitMQ : Sender using Default Exchange to send message to Queue



Objects

The following specific objects are displayed in CAST Enlighten:

Icons	Description
	• ActiveMQ Producer Queue Call • IBM MQ Producer Queue Call • RabbitMQ Exchange
	• ActiveMQ Consumer Queue Receive • IBM MQ Consumer Queue Receive • RabbitMQ Queue

Links

For **ActiveMQ**, Call link is created between:

- Producer method object and Queue Call object, at the analyser level

- Consumer method object and Queue Receive object, at the analyser level
- Queue Call object and Queue Receive object, at the Application level by Web Services Linker

For **IBM MQ**, Call link is created between:

- Producer method object and Queue Call object, at the analyser level
- Consumer method object and Queue Receive object, at the analyser level
- Queue Call object and Queue Receive object, at the Application level by Web Services Linker

For **RabbitMQ**, Call link is created between:

- Producer method object and RabbitMQ Exchange object, at the analyser level
- Consumer method object and RabbitMQ Queue object, at the analyser level/application level
- RabbitMQ Exchange object and RabbitMQ Queue object, at the application level by Web Services Linker

ActiveMQ

Example of ActiveMQ Producer(Spring-XML)

```
 <bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">
 <constructor-arg index="0" value="queue_temp"></constructor-arg>
</bean>

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
 <property name="connectionFactory" ref="jmsFactory"></property>
 <property name="defaultDestination" ref="destination" />
</bean>

<bean id="producer" class="me.andycheung.dev.springjms.producer.TestProducer">
 <property name="jmsTemplate" ref="jmsTemplate"/>
</bean>

<bean id="consumer" class="me.andycheung.dev.springjms.consumer.TestConsumer">
 <property name="jmsTemplate" ref="jmsTemplate"></property>
</bean>
```

Example of ActiveMQ Producer (Springboot)

```
        public class OrderSender {
                        private static Logger log = LoggerFactory.getLogger(OrderSender.class);
                        public static final String ORDER_QUEUE = "Sender_Queue";
                         @Autowired
                            private JmsTemplate jmsTemplate;
                            public void send(Order myMessage) {
                                log.info("sending with convertAndSend() to queue <" + myMessage + ">");
                                jmsTemplate.convertAndSend(ORDER_QUEUE, myMessage); } }
```

Example of ActiveMQ Consumer (Springboot)

```
       public class OrderConsumer {
                       private static Logger log = LoggerFactory.getLogger(OrderConsumer.class);
                           Order received;
                           private CountDownLatch countDownLatch;
                           @JmsListener(destination = Consumer_QUEUE)
                           public void receiveMessage(@Payload Order order,@Headers MessageHeaders headers,
                                                      Message message, Session session){
                                                            received = order;
                                                            log.info("received <" + order + ">"); } }
```

Example of ActiveMQ- JNDI is used to store Queue

```
  public QBorrower() throws NamingException, JMSException {
          Context ctx=new InitialContext();
          QueueConnectionFactory connectionFactory=(QueueConnectionFactory)ctx.lookup("ConnectionFactory");
          queueConnection=connectionFactory.createQueueConnection();
          requestQueue=(Queue)ctx.lookup("jms.LoanRequestQueue");
          responseQueue=(Queue)ctx.lookup("jms.LoanResponseQueue");
          queueConnection.start();
          queueSession=queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
         }

 private void sendLoanRequest(double salary,double loanAmount) throws JMSException {
          MapMessage message=queueSession.createMapMessage();
          message.setDoubleProperty("salary", salary);
          message.setDoubleProperty("loanAmount", loanAmount);
          message.setJMSReplyTo(responseQueue);
          QueueSender sender=queueSession.createSender(requestQueue);
              QueueReceiver queueReceiver=queueSession.createReceiver(responseQueue);
           sender.send(message);
        }
```

IBM MQ

Example of IBM MQ Producer (Spring Annotation)

Application. Properties

```
mq.hostName=MQ_SERVER_IP
mq.port=PORT
mq.queueManager=QUEUE.MANAGER.NAME
mq.CCSID=437
mq.username=mqm

mq.password=
mq.pubSubDomain=false
mq.receiveTimeout=20000

mq.myDestination=QUEUE_NAME
```

```
public class JmsQueueSender {

    private JmsTemplate102 jmsTemplate;
    //Referring to the value in the property file
    @Value("${mq.myDestination}")
    private String myDestination;

    public void simpleSend(final String message) {
        this.jmsTemplate.send(myDestination, new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage(message);
            }
        });
    }
}
```

Example of IBM MQ Producer (Plain Java)

```
public static void main(String args[]) {
                    int openOptions = MQC.MQOO_INQUIRE + MQC.MQOO_FAIL_IF_QUIESCING + MQC.MQOO_INPUT_SHARED;
                        MQQueue q =        qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",openOptions,null,null,
null);
                        MQMessage mBuf = new MQMessage();
                        MQPutMessageOptions pmo = new MQPutMessageOptions();
                        do {
                                runShow = br.readLine();
                                if (runShow.length() > 0) {
                                        mBuf.clearMessage();                // reset the buffer
                                        mBuf.correlationId = 1; // set correlationId
                                        mBuf.messageId = 1;     // set messageId
                                        mBuf.writeString(runShow);          // set actual message
                                        System.out.println("--> writing message to queue");
                                        q.put(mBuf,pmo);      // put the message out on the queue
                                        }
                                } while (runShow.length() > 0);
                        q.close();
                        qMgr.disconnect();
                        }
                } catch (MQException ex) {
                        System.out.println(
                "WMQ exception occurred : Completion code ");
                }
        }
```

Example of IBM MQ Consumer (Spring with JMS interface)

Spring.XML

```
<bean id="jmsQueueListener" class="hu.vanio.jms.spring3.ibmmq.JmsQueueListener" />

    <!-- and this is the message listener container -->
    <jms:listener-container connection-factory="jmsQueueConnectionFactory">
        <jms:listener destination="${mq.myDestination}" ref="jmsQueueListener" />
    </jms:listener-container>
```

```
public class JmsQueueListener implements MessageListener {

    AtomicInteger a = new AtomicInteger(0);

    public void onMessage(Message message) {
        a.addAndGet(1);
        System.out.println("\nIncoming message! (" + a + ")");
        if (message instanceof TextMessage) {
            try {
                System.out.println(((TextMessage) message).getText());
            } catch (JMSException ex) {
                throw new RuntimeException(ex);
            }
        } else {
            throw new IllegalArgumentException("Message must be of type TextMessage");
        }
    }

}
```

Example of IBM MQ Producer and Consumer in the same file (JMS Interface)

```
      public static void main(String[] args) {
       try {
           MQQueueConnectionFactory cf = new MQQueueConnectionFactory();
           cf.setHostName(HOST_NAME);
           cf.setPort(PORT_NUMBER);
           cf.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
           cf.setQueueManager(QUEUE_MANAGER_NAME);
           cf.setChannel("SYSTEM.DEF.SVRCONN");

           MQQueueConnection connection = (MQQueueConnection) cf.createQueueConnection(LOGIN_USERNAME,
LOGIN_PASSWORD);
           MQQueueSession session = (MQQueueSession) connection.createQueueSession(false, Session.
AUTO_ACKNOWLEDGE);
           MQQueue queue = (MQQueue) session.createQueue(QUEUE_NAME);
           MQQueueSender sender = (MQQueueSender) session.createSender((Queue) queue);
           sender.setPriority(0); //if not, default is 4
           sender.setDeliveryMode(DeliveryMode.NON_PERSISTENT); //DeliveryMode.PERSISTENT, the default
           MQQueueReceiver receiver = (MQQueueReceiver) session.createReceiver((Queue) queue);

           long uniqueNumber = System.currentTimeMillis() % 1000;
           JMSTextMessage message = (JMSTextMessage) session.createTextMessage("random number: " +
uniqueNumber);

           // Start the connection
           connection.start();

           // try to send 1 test message
           sender.send(message);
           System.out.println("Sent message:" + message);

           // and then get the first message from MQ
           JMSTextMessage receivedMessage = (JMSTextMessage) receiver.receive();
           System.out.println("Received message:" + receivedMessage);

           sender.close();
           receiver.close();
           session.close();
           connection.close();
           System.out.println("DONE");
       } catch (JMSException jmsex) {
           jmsex.printStackTrace();
       } catch (Exception ex) {
           ex.printStackTrace();
       }
   }
```

Example of IBM MQ Consumer (Plain Java)

```
private void read() throws MQException
{

  MQQueue queue = _queueManager.accessQueue( inputQName,
                                  openOptions,
                                  null,           // default q manager
                                  null,           // no dynamic q name
                                  null );         // no alternate user id

  MQGetMessageOptions getOptions = new MQGetMessageOptions();
  getOptions.options = MQC.MQGMO_NO_WAIT + MQC.MQGMO_FAIL_IF_QUIESCING + MQC.MQGMO_CONVERT;
  while(true)
  {
    MQMessage message = new MQMessage();
    try
    {
     queue.get(message, getOptions);
     byte[] b = new byte[message.getMessageLength()];
     message.readFully(b);
     System.out.println(new String(b));
     message.clearMessage();
    }
  }
  queue.close();
  _queueManager.disconnect();
}
```

RabbitMQ

Example of Spring AMQP RabbitMQ Producer

```
        @Service
        public class CustomMessageSender {
            private static final Logger log = LoggerFactory.getLogger(CustomMessageSender.class);
            private final RabbitTemplate rabbitTemplate;
            @Autowired
            public CustomMessageSender(final RabbitTemplate rabbitTemplate) {
                this.rabbitTemplate = rabbitTemplate;
            }
        @Scheduled(fixedDelay = 3000L)
        public void sendMessage() {
            final CustomMessage message = new CustomMessage("Hello there!", new Random().nextInt(50), false);
            log.info("Sending message...");
            rabbitTemplate.convertAndSend(MessagingApplication.EXCHANGE_NAME, MessagingApplication.
ROUTING_KEY, message);
        }  }
```

Example of Spring AMQP RabbitMQ Consumer

```
    @Service
    public class CustomMessageListener {
        private static final Logger log = LoggerFactory.getLogger(CustomMessageListener.class);
        @RabbitListener(queues = MessagingApplication.QUEUE_GENERIC_NAME)
        public void receiveMessage(final Message message) {
            log.info("Received message as generic: {}", message.toString());
        }
        @RabbitListener(queues = MessagingApplication.QUEUE_SPECIFIC_NAME)
        public void receiveMessageSpecific(final CustomMessage customMessage) {
            log.info("Received message as specific class: {}", customMessage.toString());
        } }
```

Example of SpringBoot RabbitMQ Exchange-Queue Binding configuration

```
public class MessagingApplication implements RabbitListenerConfigurer{
     public static final String EXCHANGE_NAME = "appExchange";
     public static final String QUEUE_GENERIC_NAME = "appGenericQueue";
     public static final String QUEUE_SPECIFIC_NAME = "appSpecificQueue";
     public static final String ROUTING_KEY = "messages.key";
     public static void main(String[] args) {
          SpringApplication.run(MessagingApplication.class, args);
     }
     @Bean
      public TopicExchange appExchange() {
          return new TopicExchange(EXCHANGE_NAME);
     }
     @Bean
     public Queue appQueueGeneric() {
          return new Queue(QUEUE_GENERIC_NAME);
     }
     @Bean
     public Queue appQueueSpecific() {
          return new Queue(QUEUE_SPECIFIC_NAME);
     }
     @Bean
     public Binding declareBindingGeneric() {
          return BindingBuilder.bind (appQueueGeneric()).to(appExchange()).with(ROUTING_KEY);
     }
     @Bean
     public Binding declareBindingSpecific() {
          return BindingBuilder.bind(appQueueSpecific()).to(appExchange()).with(ROUTING_KEY);
     }
```

Example of Spring AMQP RabbitMQ XML based configuration

```
<rabbit:template id="amqpTemplate" exchange="myExchange" routing-key="foo.bar" /> <rabbit:queue name="myQueue"
/>
 <rabbit:topic-exchange name="myExchange">
   <rabbit:bindings>
           <rabbit:binding queue="myQueue" pattern="foo.*" />
   </rabbit:bindings>
</rabbit:topic-exchange>
<rabbit:listener-container connection-factory="connectionFactory">
   <rabbit:listener ref="consumer" method="listen" queue-names="myQueue" />
</rabbit:listener-container>
<bean id="consumer" class="com.baeldung.springamqp.consumer.Consumer" />
```

Limitations

The following cases are not handled:

- When the queue name is given at the runtime i.e. when Queue name is not initialized anywhere in the code and is given dynamically during the session/connection