Simple Spring JMS

This is a very simple example using a Spring JMS Template to send messages and also having a JMS listener process the messages sent. An embedded ActiveMQ instance is used as the broker.

1. Producer Configuration

Spring Configuration

The Spring configuration shows a *context:component-scan* that picks up the JMS producer and listener. Following this the Spring custom namespace for Apache's ActiveMQ is used to create an embedded JMS broker. A queue is configured for 'org.springbyexample.jms.test'. Then a JMS connection factory is made for the `JmsTemplate` to use. The template will be used by the producer to send messages.

Excerpt from *JmsMessageListenerTest-context.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jms="http://www.springframework.org/schema/jms"
       xmlns:amq="http://activemq.apache.org/schema/core"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-
context.xsd
```

```
                            http://www.springframework.org/schema/jms
                            http://www.springframework.org/schema/jms/spring-jms.xsd
                            http://activemq.apache.org/schema/core
                            http://activemq.apache.org/schema/core/activemq-core.xsd">

    <context:component-scan base-package="org.springbyexample.jms" />

    <!--  Embedded ActiveMQ Broker -->
    <amq:broker id="broker" useJmx="false" persistent="false">
        <amq:transportConnectors>
            <amq:transportConnector uri="tcp://localhost:0" />
        </amq:transportConnectors>
    </amq:broker>

    <!--  ActiveMQ Destination  -->
    <amq:queue id="destination" physicalName="org.springbyexample.jms.test" />

    <!-- JMS ConnectionFactory to use, configuring the embedded broker using XML -->
    <amq:connectionFactory id="jmsFactory" brokerURL="vm://localhost" />

    <!-- JMS Producer Configuration -->
    <bean id="jmsProducerConnectionFactory"
          class="org.springframework.jms.connection.SingleConnectionFactory"
          depends-on="broker"
          p:targetConnectionFactory-ref="jmsFactory" />

    <bean id="jmsProducerTemplate" class="org.springframework.jms.core.JmsTemplate"
          p:connectionFactory-ref="jmsProducerConnectionFactory"
          p:defaultDestination-ref="destination" />

    ...

</beans>
```

Code Example

The producer uses `@PostConstruct` to indicate that `generateMessages()` is an initialization method. It uses the `JmsTemplate` to send text messages and also sets an `int` property for the message count.

Example 1. **`JmsMessageProducer`**

src/main/java/org/springbyexample/jms/JmsMessageProducer.java

```
@Component
public class JmsMessageProducer {

    private static final Logger logger =
LoggerFactory.getLogger(JmsMessageProducer.class);

    protected static final String MESSAGE_COUNT = "messageCount";

    @Autowired
    private JmsTemplate template = null;
    private int messageCount = 100;

    /**
     * Generates JMS messages
     */
    @PostConstruct
    public void generateMessages() throws JMSException {
        for (int i = 0; i < messageCount; i++) {
            final int index = i;
            final String text = "Message number is " + i + ".";

            template.send(new MessageCreator() {
```

```
                public Message createMessage(Session session) throws JMSException {
                    TextMessage message = session.createTextMessage(text);
                    message.setIntProperty(MESSAGE_COUNT, index);

                    logger.info("Sending message: " + text);

                    return message;
                }
            });
        }
    }

}
```

2. Client Configuration

Spring Configuration

This shows configuring the JMS listener using Springs *jms* custom namespace. The *jmsMessageListener* bean was loaded by the *context:component-scan* and implements `MessageListener`. If it didn't the *jms:listener* element could specify which method should process a message from the queue. The *jms:listener* specifies the *destination* attribute to be 'org.springbyexample.jms.test', which matches the queue defined by the *amq:queue* element in the embedded ActiveMQ configuration.

The `AtomicInteger` is used by the listener to increment how many messages it processes, and is also used by the unit test to confirm is received all the messages from the producer.

Excerpt from *JmsMessageListenerTest-context.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jms="http://www.springframework.org/schema/jms"
       xmlns:amq="http://activemq.apache.org/schema/core"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-
context.xsd
                           http://www.springframework.org/schema/jms
                           http://www.springframework.org/schema/jms/spring-jms.xsd
                           http://activemq.apache.org/schema/core
                           http://activemq.apache.org/schema/core/activemq-core.xsd">

    <context:component-scan base-package="org.springbyexample.jms" />

    ...

    <!-- JMS Consumer Configuration -->
    <bean id="jmsConsumerConnectionFactory"
          class="org.springframework.jms.connection.SingleConnectionFactory"
          depends-on="broker"
          p:targetConnectionFactory-ref="jmsFactory" />

    <jms:listener-container container-type="default"
                            connection-factory="jmsConsumerConnectionFactory"
                            acknowledge="auto">
        <jms:listener destination="org.springbyexample.jms.test"
ref="jmsMessageListener" />
    </jms:listener-container>
```

```
    <!-- Counter for consumer to increment and test to verify count -->
    <bean id="counter" class="java.util.concurrent.atomic.AtomicInteger" />

</beans>
```

Code Example

The `JmsMessageListener` implements the JMS interface `MessageListener`. The `int` property for the message count can be retrieved before casting the message to `TextMessage`. Then the message and message count are both logged.

Example 2. **`JmsMessageListener`**

src/main/java/org/springbyexample/jms/JmsMessageListener.java

```
@Component
public class JmsMessageListener implements MessageListener {

    private static final Logger logger =
LoggerFactory.getLogger(JmsMessageListener.class);

    @Autowired
    private AtomicInteger counter = null;

    /**
     * Implementation of <code>MessageListener</code>.
     */
    public void onMessage(Message message) {
        try {
            int messageCount =
message.getIntProperty(JmsMessageProducer.MESSAGE_COUNT);
```

```
            if (message instanceof TextMessage) {
                TextMessage tm = (TextMessage)message;
                String msg = tm.getText();



                logger.info("Processed message '{}'.  value={}", msg, messageCount);


                counter.incrementAndGet();
            }
        } catch (JMSException e) {
            logger.error(e.getMessage(), e);
        }
    }

}
```