JMS Control Tutorial

The Jmscontrol is an extensible control. Before a JmsControl can be used in an application, a sub-interface of the `org.apache.beehive.controls.system.jms.JmsControl` interface must be created and annotated with `@ControlExtension`.

```
@ControlExtension
public interface SampleQueue
    extends JMSControl {
...
}
```

In order for the control to work, it needs to know the destination of the messages. This is accomplished using a JNDI context. Unless otherwise specified the default initial context is used. This may be overridden by settng the `jndiContextFactory` and `jndiProviderUrl` properties, either programically using the `setJndiContextFactory()` and `setJndiProviderUrl()` setters or via the corresponding `@Destination` attributes.

The queue/topic destination is then obtained using the value of the `sendJndiName` property and a queue/topic connection is obtained using by the `jndiConnectionFactory` property. In most cases the same connection factory is used for both queues and topics. The `@Destination` sendType attribute may be used to constrain the use of the control to either a topic or a queue. By default it's value is Auto which allowes for run-time determination of whether the `sendJndiName` names a queue or a topic. By setting it to Queue or Topic a run-time check is made to see if the connection factory and destination is of the correct type.

If the JNDI context to be used (i.e. the control is running in an ejb-container (or servlet-container with a JNDI context) is known (or is the default context) and the connection-factory (e.g. `weblogic.jms.ConnectionFactory`) and queue JNDI name (e.g. `jms.SampleQueue`) is also known at development time then the extension class can be annotated with the `@Destination` annotation as shown in the example:

```
@ControlExtension
@JMSControl.Destination(sendType=JMSControl.DestinationType.Queue,sendJndiName="jms.Sam
public interface SampleQueue
    extends JMSControl {
...
}
```

Likewise, for a topic (e.g. `jms.SampleTopic`) the following file might be appropriate:

```
@ControlExtension
@JMSControl.Destination(sendType=JMSControl.DestinationType.Topic,sendJndiName="jms.Sam
public interface SampleTopic
    extends JMSControl {
...
}
```

The `sendType` attribute could be left out of these examples and the control extensions would still work.

See Extension Class Annotation for other annotations defined at the class or type level.

The extension interface can include one or more methods that send messages. These methods must have at least one unannotated parameter that corresponds to the body of the message. Other annotated parameters can defined to provide property values and other information at run-time to the message (see Extension Class Annotation for allowed annotation). The method itself can be annotated (see Extension Class Annotation for allowed annotation).

Some examples appropriate to topics and queues include:

```
/**
 * Submit an xml object (org.apache.xmlbeans) as a text message.
 * @param document the document.
 * @param type the message JMS type.
*/
public void submitXml(XmlObject document, @Type String type);

/**
 * Submit an xml object (org.apache.xmlbeans) with JMS type "xmlObject".
 * @param document the document.
 */
@Message(MessageType.Text)
@Type("xmlObject")
public void submitXml(XmlObject document);

/**
 * Submit an already constructed message
 * @param message the jms-message.
*/
public void submitMessage(Message message);

/**
 * Submit a BytesMessage with the given byte array body and property hello.
 * @param body the byte array.
 * @param hello the value of the hello property.
*/
public void submitMessage(byte[] body, @Property(name=hello) hello);
```

```
/**
 * Submit a MapMessage with the given map and property hello set to world.
 * @param body the byte array.
*/
@Properties({PropertyValue(name="hello", value="world")})
public void submitMessage(Map body);
```