Mutual Authentication For ActiveMQ with Spring

Need to setup mutual (2-way, client, etc) SSL authentication between a command line started ActiveMQ broker and a Spring application JMS client (JmsTemplate/Apache Camel). Also, find a solution when the broker is setup in a Spring context file as part of a JUnit test.

Creating the necessary Keys and Stores

There's a great resource on creating the necessary files on the Apache site essentially you want to follow these steps:

1. Using keytool, create a certificate for the broker:
   `keytool -genkey -alias broker -keyalg RSA -keystore broker.ks`
2. Export the broker's certificate so it can be shared with clients:
   `keytool -export -alias broker -keystore broker.ks -file broker_cert`
3. Create a certificate/keystore for the client:
   `keytool -genkey -alias client -keyalg RSA -keystore client.ks`
4. Create a truststore for the client, and import the broker's certificate. This establishes that the client "trusts" the broker:
   `keytool -import -alias broker -keystore client.ts -file broker_cert`
5. Export the client's certificate so it can be shared with broker:
   `keytool -export -alias client -keystore client.ks -file client_cert`
6. Create a truststore for the broker, and import the client's certificate. This establishes that the broker "trusts" the client:
   `keytool import -alias client -keystore broker.ts -file client_cert`

ActiveMQ Broker (Started from the cmd line)

1. Before starting the broker's VM set the SSL_OPTS environment variable so that it knows to use the broker keystore.
   ```
   export SSL_OPTS = -Djavax.net.ssl.keyStore=/path/to/broker.ks -
   Djavax.net.ssl.keyStorePassword=password -
   Djavax.net.ssl.trustStore=/path/to/broker.ts
   ```
2. Start the broker from the cmd line as normal

ActiveMQ Broker (Embedded in Spring Context)

```
<beans
      xmlns="http://www.springframework.org/schema/beans"
      xmlns:amq="http://activemq.apache.org/schema/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://activemq.apache.org/schema/core http://activemq.apache.org/schem
a/core/activemq-core.xsd">

      <!--  lets create an embedded ActiveMQ Broker -->

      <amq:broker useJmx="false" persistent="true" useShutdownHook="true">
            <amq:persistenceAdapter>
                  <amq:amqPersistenceAdapter directory="data/activemq" />
                  </amq:persistenceAdapter>
                  <amq:sslContext>
                        <amq:sslContext
```

```
                        keyStore="/server/broker.ks"
keyStorePassword="Passw0rd"
                        trustStore="/server/broker.ts"
trustStorePassword="Passw0rd"/>
                  </amq:sslContext>
                  <amq:transportConnectors>
                        <amq:transportConnector name="ssl"
uri="ssl://localhost:1616?needClientAuth=true" />
                  </amq: transportConnectors>
            </amq:broker>
      </beans>
```

Enabling SSL mutual authentication in a Spring Context (JMS Client)

The steps I've described above have been fairly well documented on the internet. The following steps are where the internet starts to fail. There seems to be 2 methods to setup an SSL enabled JMS client deployed in a container

1. Set the necessary environment options before starting the container (see SSL_OPTS above)
2. Or define a custom connection factory bean see the Fuse Source Example which I'm not going to re-explain here – though be sure to add the keystore/keystorepassword to the xml example:

```
<bean id="connectionFactory"
class="com.nagarro.jms.queue.common.CustomActiveMQSslConnectionFactory">
      <property name="brokerURL" value="ssl://localhost:1616" />
      <property name="trustStore" value="/client/client.ts" />
      <property name="trustStorePassword" value="Passw0rd" />
      <property name="keyStore" value="/client/client.ks" />
      <property name="keyStorePassword" value="Passw0rd" />
</bean>
```