servicemix-script

ServiceMix Script

The ServiceMix Script component is deprecated. Please use the ServiceMix Scripting component instead.

The ServiceMix Script component provides JBI integration with scripting engines.

This component leverages Spring support for dynamic languages, so you will find useful informations on the Spring site.

Maven Archetype

You can use Maven servicemix-script-service-unit archetype to create Script service unit:

```
mvn archetype:create \
  -DarchetypeGroupId=org.apache.servicemix.tooling \
  -DarchetypeArtifactId=servicemix-script-service-unit \
  -DarchetypeVersion=2010.01 \
  -DgroupId=your.group.id \
  -DartifactId=your.artifact.id \
  -Dversion=your-version
```

Endpoint Configuration

The Script endpoint is able to run Groovy script to process messages.

xbean.xml

```
<beans xmlns:script="http://org.apache.servicemix/script/1.0"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:test="urn:test">
  <script:exchangeProcessor service="test:groovy" endpoint="endpoint">
    <property name="helpers">
      <list>
        <ref bean="groovyExchangeHelper" />
      </list>
    </property>
    <property name="implementation" ref="groovyExchangeProcessor" />
  </script:exchangeProcessor>
  <script:exchangeHelper id="groovyExchangeHelper" singleton="true" />
  <lang:groovy id="groovyExchangeProcessor"
               script-source="classpath:GroovyExchangeProcessor.groovy">
    <lang:property name="exchangeHelper" ref="groovyExchangeHelper" />
  </lang:groovy>
</beans>
```

GroovyExchangeProcessor.groovy

```
import org.apache.servicemix.common.ExchangeProcessor;
import javax.jbi.messaging.MessageExchange;
import javax.jbi.messaging.NormalizedMessage;
import org.apache.servicemix.jbi.jaxp.StringSource;
import org.apache.servicemix.script.ScriptExchangeHelper;
import org.apache.servicemix.jbi.jaxp.SourceTransformer;

class GroovyExchangeProcessor implements ExchangeProcessor {

    @Property ScriptExchangeHelper exchangeHelper;

        def void start() {
            println "Starting";
        }

    def void process(MessageExchange exchange) {
            def inputMessage = new SourceTransformer().toString(exchange.getInMessage().getContent());
            println "Hello, I got an input message "+inputMessage;
            NormalizedMessage out = exchange.createMessage();
        out.setContent(new StringSource("<world>hello</world>"));
        exchange.setMessage(out, "out");
        exchangeHelper.sendExchange(exchange);
    }

    def void stop() {
            println "Stopping";
    }
}
```

Inlining script

Alternatively, you can inline the script in the `xbean.xml` file the following way:

```
<lang:groovy id="groovyExchangeProcessor">
  <lang:inline-script><![CDATA[
      ... put your script here ...
  ]]></lang:inline-script>
  <lang:property name="exchangeHelper" ref="groovyExchangeHelper" />
</lang:groovy>
```

State and threading model

The script defines a class that will be used to process incoming exchanges, but only a single instance of this class will be used. This means that you need to make sure the script is thread safe and can be used to process requests concurrently. On the other side, this means that maintaining state is just a matter of defining a property on the class and using it to read / store any state that will be maintained.

Script Helpers

In order to create Message Exchanges one needs the Delivery Channel to obtain the Message Exchange Factory to create them. This is not possible by default but can be easily added using a customer Script Helper class. The only thing to do is to create a class that implements the **Script Helper** interface and use the **Script Exchange Processor Endpoint** instance to obtain the Delivery Channel from. Now you can either provide the Delivery Channel as is or you can create convenience method to create Message Exchanges for example. This is how a customer Script Helper class would look like:

foo.MyScriptHelper.java

```
package foo;

import javax.jbi.messaging.DeliveryChannel;

import org.apache.servicemix.script.ScriptExchangeProcessorEndpoint;
import org.apache.servicemix.script.ScriptHelper;

public class MyScriptExchangeHelper
   implements ScriptHelper
{
    protected ScriptExchangeProcessorEndpoint mEndpoint;

    public void setScriptExchangeProcessorEndpoint(
        ScriptExchangeProcessorEndpoint pEndpoint
    ) {
        mEndpoint = pEndpoint;
    }

    public DeliveryChannel getChannel() {
        return mEndpoint.getChannel();
    }
}
```

After that you only need to use a Spring Bean to create an instance and the use the same references in the rest of the Script settings. This is what needs to be changed from the **xbean.xml** mentioned above:

xbean.xml changes

```
<bean
    id="groovyExchangeHelper"
    class="foo.MyScriptExchangeHelper"
/>

<!--<script:exchangeHelper id="groovyExchangeHelper" singleton="true" />-->
```

The Groovy code could then look like this:

GroovyExchangeProcessor.groovy

```
import org.apache.servicemix.common.ExchangeProcessor;
import javax.jbi.messaging.MessageExchange;
import org.apache.servicemix.jbi.jaxp.StringSource;
import org.apache.servicemix.script.ScriptHelper;
import org.apache.commons.logging.LogFactory;

class GroovyExchangeProcessor
    implements ExchangeProcessor
{
    @Property ScriptHelper exchangeHelper;

    def mLog = LogFactory.getLog( getClass() );

    def void start() {
        mLog.info( "Starting" );
    }

    def void process(MessageExchange pExchange) {
        def lChannel = exchangeHelper.getChannel();
        def lRequest = lChannel.createExchangeFactory().createInOutExchange();
        lRequest.setService( new QName( "urn:test", "target-service" ) );
        lRequest.setOperation( new QName( "MyOpertation" ) );
        def lNM = lRequest.createMessage();
        lNM.setContent( new StringSource( "<receiver><title>Zero</title><index>0</index></receiver>" ) );
        lRequest.setInMessage( lNM );
        mLog.info( "process() send new ME: " + lRequest );
        lChannel.send( lRequest );
    }

    def void stop() {
        mLog.info( "Stopping" );
    }
}
```

Sending exchanges

When acting as a **Consumer** (i.e. creating and sending an exchange into the bus), this Endpoint is currently limited to sending **InOnly** requests. This limitation is solved for versions >= 3.2.2

Tips when creating Script Helpers

- When defining your own helper, if you do not extend the **ScriptExchangeHelper** then you will need to change the type in the Groovy script to **ScriptHelper** for the **exchangeHelper** propery
- if you extend the **ScriptExchangeHelper** then you have to call the super.setScriptExchangeProcessorEndpoint() method in order to set the private member there too, otherwise you will experience *Null Pointer Exceptions*
- to use your own helper, you have to use the Spring **<bean />** element in the xbean.xml as shown in the above example because the *script:exchangeHelper* is fixed to use the `ScriptExchangeHelper` class
- as Groovy is a dynamic language, there is no need to upcast the exchange helper property in the Groovy script: even though the `ScriptHelper` type is specified in the script, Groovy will be able to find the methods in the sub class by introspecting the object