

Grails + CXF Example

Grails + CXF Example

Ok, we've been trying to get full fledged web services (providing) working in Grails for quite a while, it felt like it should be easy, as all the bits were there, but all of the plugins to date were more proof of concept vs Enterprise ready (e.g. no control over namespaces).

So, the target was to get CXF running with the following key requirements:

- WSDL First
- Minimal change required to CXF generated interface classes
- Bound to Grails services so we can re-use business logic and interact with domain classes

Step 1: Grails App + CXF

Create a simple Grails application (working from the ground up is the best way to explain it, rather than trying to explain our existing application), create a simple domain class, controller and service.

Download Apache CXF (I used 2.1): <http://cxf.apache.org/> and extract somewhere on your PC.

Copy the following libraries from the CXF Installation over into your Grails application lib folder:

```
XmlSchema-1.4.2.jar
commons-logging-1.1.1.jar
cxf-2.1.jar
cxf-manifest.jar
geronimo-activation_1.1_spec-1.0.2.jar
geronimo-annotation_1.0_spec-1.1.1.jar
geronimo-javamail_1.4_spec-1.3.jar
geronimo-jms_1.1_spec-1.1.1.jar
geronimo-servlet_2.5_spec-1.2.jar
geronimo-stax-api_1.0_spec-1.0.1.jar
geronimo-ws-metadata_2.0_spec-1.1.2.jar
jaxb-api-2.1.jar
jaxb-impl-2.1.6.jar
jaxb-xjc-2.1.6.jar
jaxws-api-2.1-1.jar
neethi-2.0.4.jar
saaj-api-1.3.jar
saaj-impl-1.3.jar
wstx-asl-3.2.4.jar
xml-resolver-1.2.jar
asm-2.2.3.jar
stax-utils-20060502.jar
jaxen-1.1.jar
jdom-1.0.jar
```

Next, add a resources.xml file to your grails-app/conf/Spring folder:

```
<beans
xmlns="http://www.springframework.org/schema
/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
  xmlns:simple="http://cxf.apache.org/simple"
  xmlns:lang="http://www.springframework.org/
schema/lang"

xmlns:jaxws="http://cxf.apache.org/jaxws"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/
spring-beans.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/s
pring-lang-2.0.xsd
http://cxf.apache.org/simple
http://cxf.apache.org/schemas/simple.xsd
http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd">

  <import
resource="classpath:META-INF/cxf/cxf.xml" />
  <import
resource="classpath:META-INF/cxf/cxf-extensi
on-soap.xml" />
  <import
resource="classpath:META-INF/cxf/cxf-servlet
.xml" />
</beans>
```

Next, install templates for your app (grails install-templates), then go into src/templates/war and edit web.xml, add the following:

```
<servlet>
  <servlet-name>CXFServlet</servlet-name>
  <display-name>CXF Servlet</display-name>
  <servlet-class>

org.apache.cxf.transport.servlet.CXFServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>CXFServlet</servlet-name>
  <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

Next, ensure that you turn off the inbuilt URL Mapping for the /ws/* in conf/UrlMapping.groovy:

```

class UrlMappings {
    static mappings = {
        "/$controller/$action?/$id?" {
            constraints {
                // apply constraints here
            }
        }
        controller(matches: /\.*/[^(ws)].*/)
    }
}
"500"(view: '/error')
}
}

```

Ok, now you should be able to start up your application (do this just to make sure you haven't broken anything).

Step 2: WSDL First CXF Service

Now, grab a WSDL, in this case I will use [currencyConverterService.wsdl](#) that comes with one of the many services frameworks I downloaded recently. Go into the bin folder of the Apache CXF folder (where you saved it), and run the following command:

```

wsdl2java -d <outputFolder> -impl
<pathtoWSDL>/currencyConverterService.wsdl

```

This generates an interface, an implementation and classes for all of the types - all properly marked up and configured. Copy everything generated into the src/java folder in your Grails app (from the com folder up).

Now comes the fun bit!

Step 3: Bind the CXF Service into your Grails Application

Ok, now what we need to do is to tell our Grails application that we have a service, and more importantly bind that service into an existing Grails service.

First we need to edit our Grails service, so that it implements the interface generated by CXF, and then add the methods of that interface with some code:

```

class ConverterService implements
com.examplewebservice.CurrencyConvert {

com.examplewebservice.types.ConversionRespon
se convert(

com.examplewebservice.types.ConversionReques
t part1
    ) {
        def resp = new
com.examplewebservice.types.ConversionRespon
se()
        // We could do interesting things
here, get a domain class etc.
        // Just return something
resp.amount = 123.00
return resp
    }
}

```

Ok, good so far. Next step is to do a very small edit to the CXF Service Implementation to do the following things:

1. We need to add a placeholder so that we can get Spring to inject the Grails Service into this class so that we can pass through the requests from the CXF class to the Grails service.
2. Add code to the service method so that it passes through to the Grails Service

The complete code is below:

```

/**

```



```
        targetNamespace =
"http://www.examplewebservice.com",
        wsdlLocation =
"file:test/currencyConverterService.wsdl",
        endpointInterface =
"com.examplewebservice.CurrencyConvert")
```

```
public class CurrencyConvertImpl implements
CurrencyConvert {
```

```
    // CHANGE 1. HERE
```

```
    // Link to Groovy Object - this is
required for all Grails service
implementations
```

```
    private CurrencyConvert groovyObject;
```

```
    public void
```

```
setGroovyObject(CurrencyConvert
groovyInstance) { this.groovyObject =
groovyInstance; }
```

```
    private static final Logger LOG =
Logger.getLogger(CurrencyConvertImpl.class.g
etName());
```

```
    /* (non-Javadoc)
```

```
    * @see
```

```
com.examplewebservice.CurrencyConvert#conver
t(com.examplewebservice.types.ConversionRequ
est part1 )*
```

```
    */
```

```
    public
```

```
com.examplewebservice.types.ConversionResponse
convert(com.examplewebservice.types.ConversionRequest part1) {
    LOG.info("Executing operation
convert");
    System.out.println(part1);
    try {
        // CHANGE 2. HERE
        // pass through requests to the
Groovy/Grails Service
```

```
com.examplewebservice.types.ConversionResponse _return = groovyObject.convert(part1);
    return _return;
} catch (Exception ex) {
    ex.printStackTrace();
    throw new RuntimeException(ex);
}
```

```
}  
  
}
```

Next, we need to go back to our resources.xml and tell CXF that we now have a service, and provide the magic that injects the Grails service into the CXF one to link them together (add the following code inside the <beans> element of the file, under the CXF imports:

```
    <!--create the bean for the service,  
link to groovy service bean -->  
    <bean id="currencyConverter"  
class="com.examplewebservice.CurrencyConvert  
Impl">  
        <property name="groovyObject"  
ref="converterService" />  
    </bean>  
  
    <!--create CXF service-->  
    <simple:server  
serviceClass="com.examplewebservice.Currency  
Convert" address="/ConverterService">  
        <simple:serviceBean>  
            <ref bean="currencyConverter" />  
        </simple:serviceBean>  
    </simple:server>
```

The key thing here is the bean creation, that injects the link to the Groovy Service (which is just another Bean once Grails is running) into the groovyObject property of our CXF service. The other piece is just the simple method of creating a CXF service.

Note:

It is possible (in some cases - I haven't figured out why it works sometimes), to not use the intermediate Java Impl class, but rather configure the CXF Service to point straight at the Grails Service bean (provided it implements the interface). This makes it much faster, as you don't need to change the generated code at all (ideal).

In this case, the resources.xml would not have the `<bean ...>` and `<ref bean="currencyConverter" />` becomes `<ref bean="converterService"/>`. Further exploration may explain why it works sometimes but not others (I get a strange error during startup of a Null parameter to CXF).

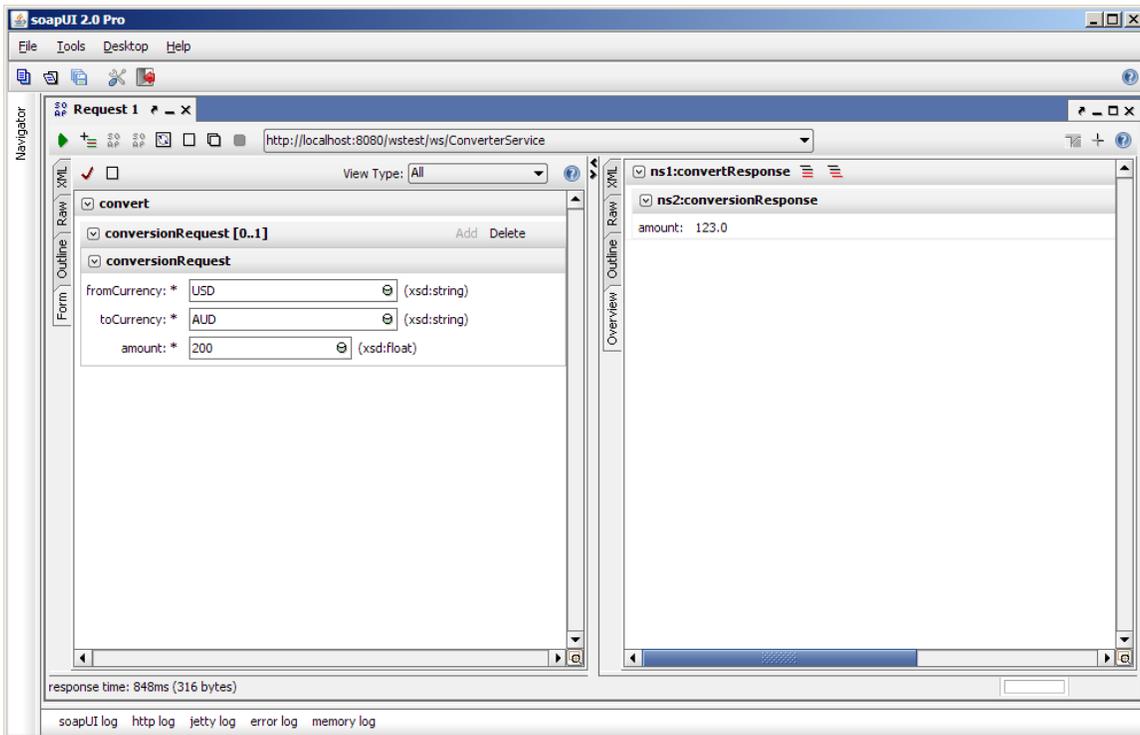
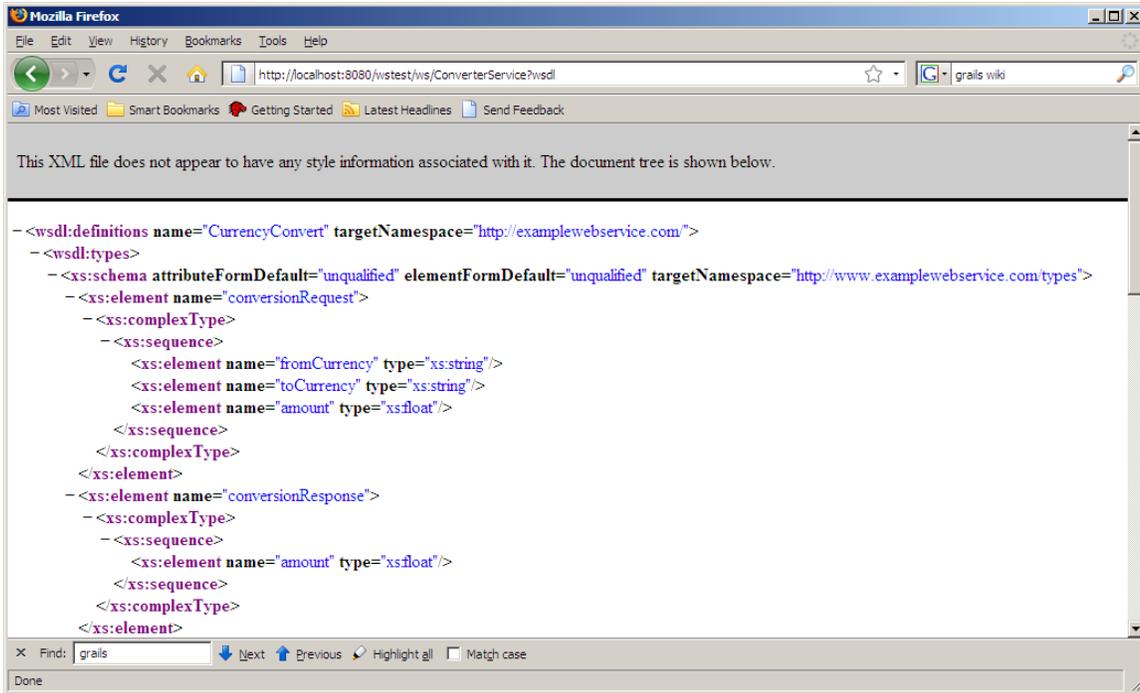
Step 4: Test!

When you run the application, you should see something like this appearing in the log:

```
01-Jun-2008 17:07:52
org.apache.cxf.service.factory.ReflectionSer
viceFactoryBean buildServiceFromClass
INFO: Creating Service
{http://examplewebservice.com/}CurrencyConve
rt from class
com.examplewebservice.CurrencyConvert
01-Jun-2008 17:07:52
org.apache.cxf.endpoint.ServerImpl
initDestination
INFO: Setting the server's publish address
to be /ConverterService
01-Jun-2008 17:07:53
org.apache.cxf.transport.servlet.CXFServlet
loadSpringBus
INFO: Load the bus with application context
01-Jun-2008 17:07:53
org.apache.cxf.bus.spring.BusApplicationCont
ext getConfigResources
INFO: No cxf.xml configuration file
detected, relying on defaults.
01-Jun-2008 17:07:53
org.apache.cxf.transport.servlet.AbstractCXF
Servlet replaceDestinationFactory
INFO: Servlet transport factory already
registered
```

Then, when you browse to: <http://localhost:8080/wstest/ws/services> you should get the list of services (in this case just one), and then be able to view the generated WSDL (which should have all the same namespaces etc. as the original).

Here are some pictures to prove that it works 😊



Next ... ?

- No idea if this is the best way to do it, or if it will cause issues later on with respect to trying to use any of the more complex features of CXF.
- Also haven't done much testing to make sure it doesn't break other plugins, and we can still deploy it to OC4J with all the new libraries

Clearly, any comments or feedback welcome!