Brians Java Blog

MONDAY, 14 JANUARY 2013

Spring Web Services Tutorial

Introduction

Modern enterprise applications are rarely stand alone and often rely on data and services provided by external systems. In order for different types of systems to communicate there must be a communication protocol of some sort, a standard way of sending and receiving messages in a format that is recognised and supported by all major platforms. SOAP (Simple Object Application Protocol) is such a protocol, and allows applications to communicate by exchanging messages in a standard XML format.

SOAP Web Services provide a platform agnostic integration mechanism that allows disparate systems to exchange data regardless of the platform they are running on. For example, SOAP web services are commonly used to integrate .NET applications with applications running on the Java platform. Almost all modern platforms and frameworks (Java, .Net, Ruby, PHP, etc) provide comprehensive libraries and tooling that allow developers to quickly and easily expose and consume SOAP services.

This post will look at Spring Web Services and take you through a step by step tutorial for building, deploying and testing a simple contract first SOAP service for retrieving simple bank account details.

Technology Stack

The technology stack used in this tutorial will include Spring 3.1 for Web Services Support, Maven for dependency resolution & build, Tomcat for our test server and SoapUI to build sample SOAP messages for testing our service.

BLOG ARCHIVE

- ► 2014 (3)
- ▼ 2013 (6)
  - ► October (1)
  - ► April (2)
  - ► March (1)
  - ▼ January (2)
    - Axis2 Web Service Client Tutorial
    - Spring Web Services Tutorial
- ► 2012 (10)

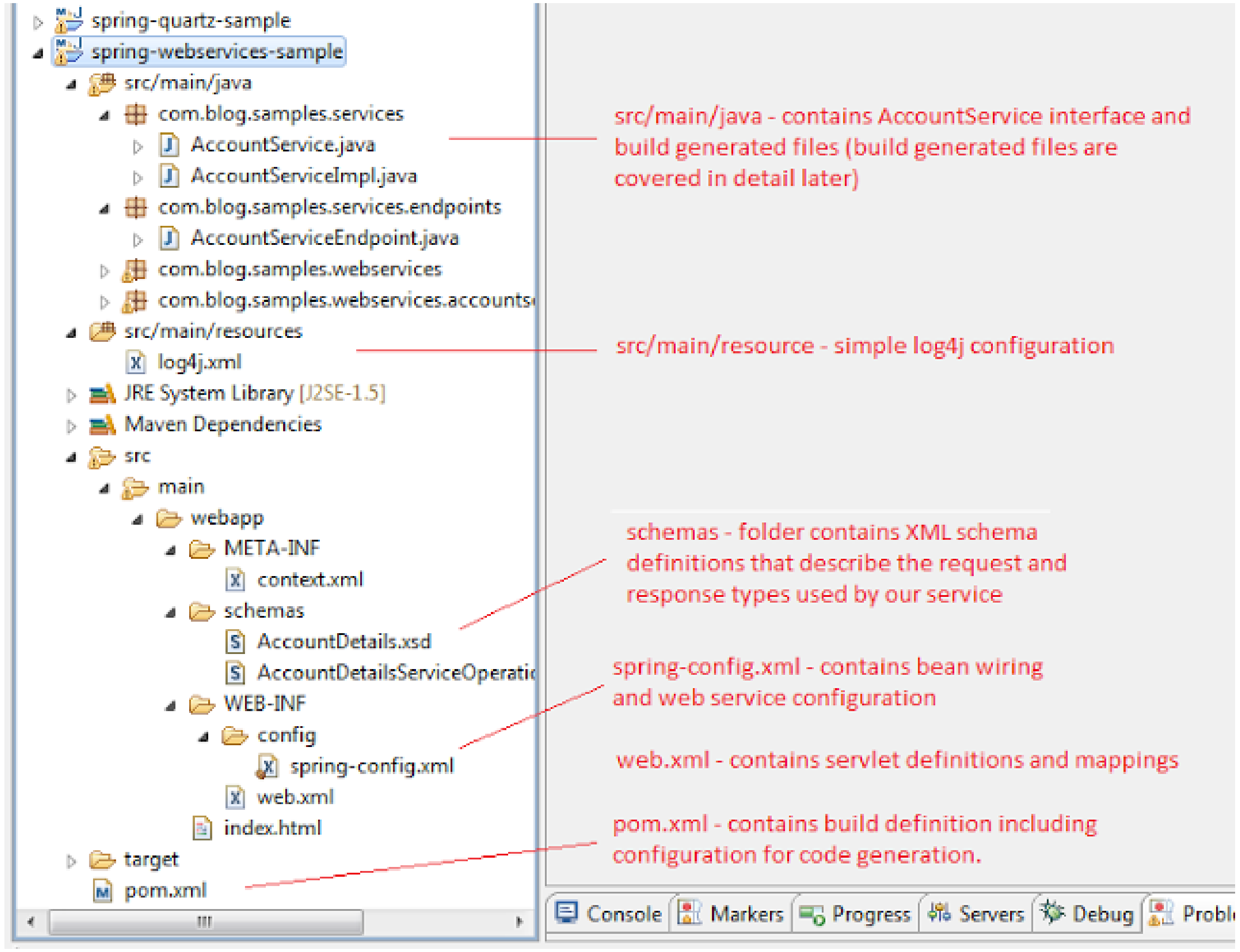
ABOUT ME

Brian

View my complete profile

Creating the Project

The project structure is similar to that used in a some of my other tutorials and is typical of most modern Spring web applications. We'll start off by creating a simple Spring web project like the one shown in figure 1.0 below.



```
spring-quartz-sample
spring-webservices-sample
  src/main/java
    com.blog.samples.services
      AccountService.java
      AccountServiceImpl.java
    com.blog.samples.services.endpoints
      AccountServiceEndpoint.java
    com.blog.samples.webservices
    com.blog.samples.webservices.accounts
  src/main/resources
    log4j.xml
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  src
    main
      webapp
        META-INF
          context.xml
        schemas
          AccountDetails.xsd
          AccountDetailsServiceOperati
        WEB-INF
          config
            spring-config.xml
          web.xml
        index.html
  target
  pom.xml
```

src/main/java - contains AccountService interface and build generated files (build generated files are covered in detail later)

src/main/resource - simple log4j configuration

schemas - folder contains XML schema definitions that describe the request and response types used by our service

spring-config.xml - contains bean wiring and web service configuration

web.xml - contains servlet definitions and mappings

pom.xml - contains build definition including configuration for code generation.

Figure 1.0 Project Structure

Contract Last vs Contract First

There are two fundamental approaches to building web services, Contract Last and Contract First. The Contract Last approach involves taking existing code and generating a service contract directly from

that code in order to expose it as a SOAP interface. There are a variety of Java frameworks out there (Axis2, XFire etc) that provide this Java2WSDL tooling, to quickly generate the server side proxies, marshallers and Servlet classes required to expose a SOAP service.
The Contract First approach involves defining the Service contract before implementing the service. This means describing the service parameters and return types using XSD's (XML Schema Definitions), then using those XSD's to construct a WSDL (web service definition language) to provides a public facing contract or description of the service. Only after the service contract has been clearly defined, is the service implementation actually written.
This post will describe a Contract First service, as this is the preferred approach for various reasons, some of which are explained in this article.

Service Contract Definition

Given that we're building a service to retrieve simple bank account details we'll start off by defining our core business entity, an Account. We'll define our account entity in src/main/webapp/schemas/AccountDetails.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://webservices.samples.blog.c
     <xs:element name="Account" type="Account"/>
     <xs:complexType name="Account">
          <xs:sequence>
               <xs:element name="AccountNumber" type="xs:string"/>
               <xs:element name="AccountName" type="xs:string"/>
               <xs:element name="AccountBalance" type="xs:double"/>
               <xs:element name="AccountStatus" type="EnumAccountStatus"/>
          </xs:sequence>
     </xs:complexType>
     <xs:simpleType name="EnumAccountStatus">
          <xs:restriction base="xs:string">
               <xs:enumeration value="Active"/>
               <xs:enumeration value="Inactive"/>
          </xs:restriction>
     </xs:simpleType>
```

```
</xs:schema>
```

I use XMLSpy for working with XML as it provides a useful graphical representation of the XML types being defined. This can be useful when working on large applications with complex data models. A visual representation of the above XSD is shown below.

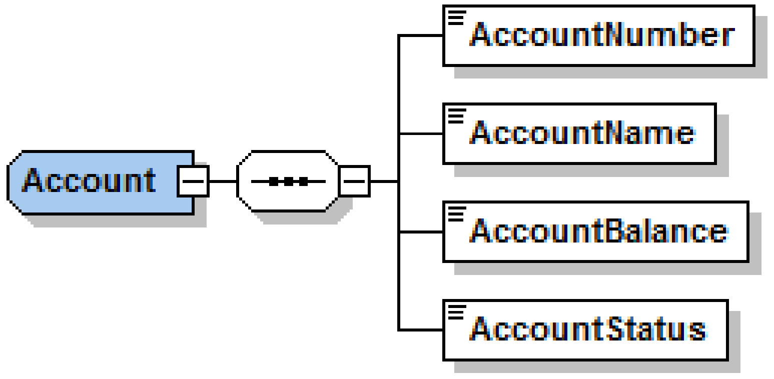


Figure 2.0 Account Entity

Next we'll define the service request and response types in src/main/webapp/schemas/AccountDetailsServiceOperations.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://com/blog/samples/webserv
     <xsd:import namespace="http://webservices.samples.blog.com" schemaLocation="AccountDetails
     <xsd:element name="AccountDetailsRequest">
          <xsd:complexType>
               <xsd:sequence>
                    <xsd:element name="accountNumber" type="xsd:string"/>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:element>
     <xsd:element name="AccountDetailsResponse">
          <xsd:complexType>
```

```
                <xsd:sequence>
                     <xsd:element name="AccountDetails" type="account:Account"/>
                </xsd:sequence>
           </xsd:complexType>
      </xsd:element>
</xsd:schema>
```

A visual representation of these types is shown below.

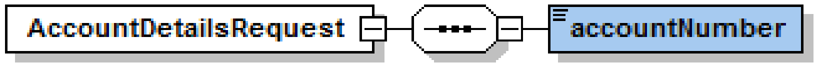


Figure 3.0 AccountDetailsRequiest Entity

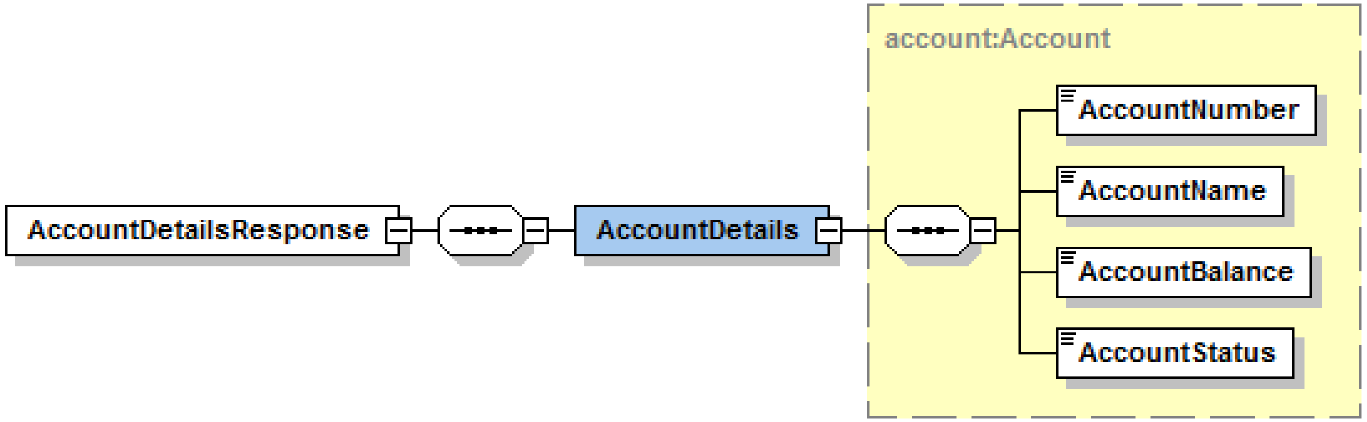


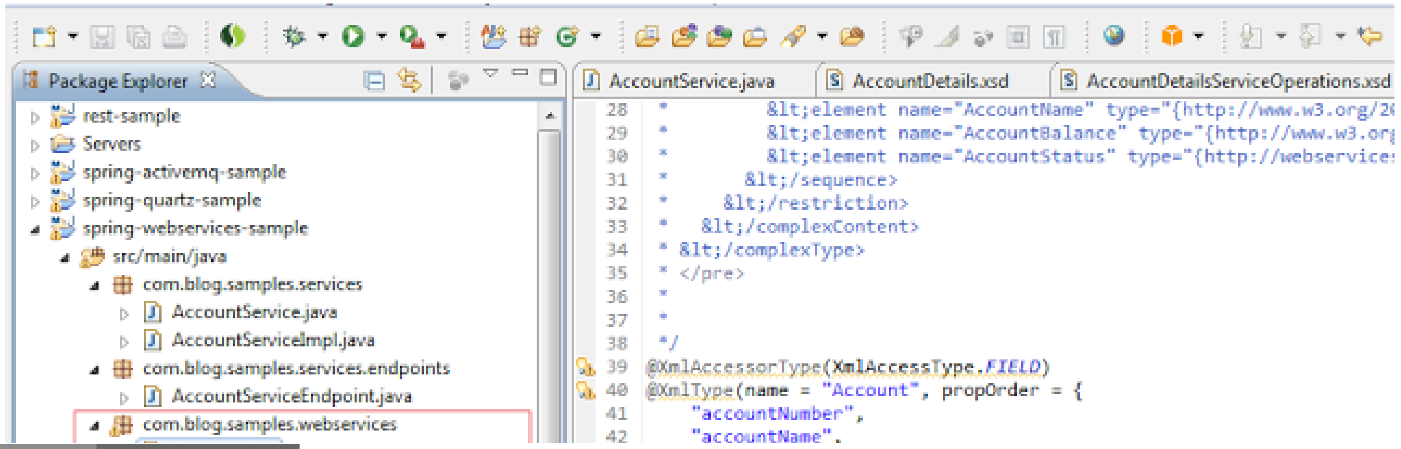
Figure 4.0 AccountDetailsResponse Entity

Object to XML Mapping

A fundamental part of web services is the conversion of SOAP messages from XML to Java objects and vice versa. This is a non trivial task if you were to set out to do it yourself so we'll make use of the JAXB framework to take car of this for us. I've been working with JAXB for a few years now and find it to be a powerful and flexible framework, and a huge improvement on older OXM frameworks like Castor.
In order to use our XSD defined types in the application we need to generate Java classes from those types. We do this as part of the Maven build process by using the jaxb-maven-plugin in our POM. The plugin is configured to parse a set of XSD's and run JAXB's class generator to create Java classes for each of the defined types. For brevity only part of the Maven POM definition is shown below. The entire POM definition can be found with the source code that accompanies this tutorial.

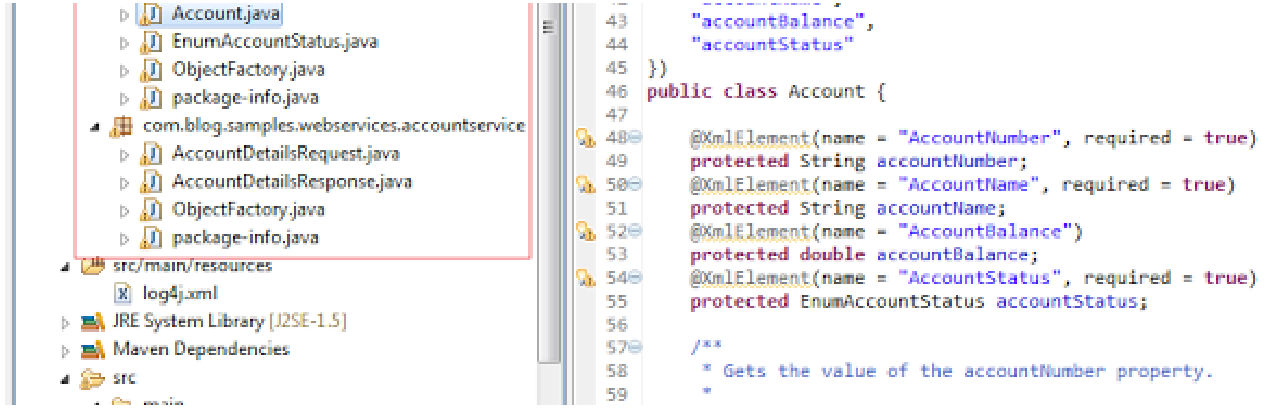
```xml
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                                         http://maven.apache.org/maven-v4_0_0.xsd">
     <artifactId>spring-webservices-sample</artifactId>
     <modelVersion>4.0.0</modelVersion>
     <inceptionYear>2013</inceptionYear>
     <packaging>war</packaging>
     <groupId>com.blog.webservices</groupId>
     <version>1.0</version>
     <properties>
          <spring.version>3.1.1.RELEASE</spring.version>
          <spring.ws.version>2.0.0.RELEASE</spring.ws.version>
          <log4j.version>1.2.16</log4j.version>
          <context.path>spring-webservices-sample</context.path>
     </properties>
     <build>
          <plugins>
               <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>jaxb2-maven-plugin</artifactId>
                    <version>1.4</version>
                    <executions>
                         <execution>
                              <goals>
                                   <goal>xjc</goal>
                              </goals>
                              <phase>generate-sources</phase>
                         </execution>
                    </executions>
                    <configuration>
                    <clearOutputDir>false</clearOutputDir>
                    <outputDirectory>src/main/java</outputDirectory>
                    <schemaDirectory>src/main/webapp/schemas</schemaDirectory>
                    <includeSchema>**/*.xsd</includeSchema>
                    <enableIntrospection>false</enableIntrospection>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <warName>${context.path}</warName>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
...
...
```

Running a Maven build will create Java classes for each of the defined schema types. The screenshot below shows what the generated classes should look like in your project after you run a Maven build. Note that JAXB has used the the namespaces in the XSD's to derive package names for the generated classes.



```
rest-sample
Servers
spring-activemq-sample
spring-quartz-sample
spring-webservices-sample
  src/main/java
    com.blog.samples.services
      AccountService.java
      AccountServiceImpl.java
    com.blog.samples.services.endpoints
      AccountServiceEndpoint.java
    com.blog.samples.webservices
```

```
 *           &lt;element name="AccountName" type="{http://www.w3.org/2
 *           &lt;element name="AccountBalance" type="{http://www.w3.or
 *           &lt;element name="AccountStatus" type="{http://webservice
 *         &lt;/sequence>
 *       &lt;/restriction>
 *     &lt;/complexContent>
 *   &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Account", propOrder = {
    "accountNumber",
    "accountName",
```



```
    "accountBalance",
    "accountStatus"
})
public class Account {

    @XmlElement(name = "AccountNumber", required = true)
    protected String accountNumber;
    @XmlElement(name = "AccountName", required = true)
    protected String accountName;
    @XmlElement(name = "AccountBalance")
    protected double accountBalance;
    @XmlElement(name = "AccountStatus", required = true)
    protected EnumAccountStatus accountStatus;

    /**
     * Gets the value of the accountNumber property.
     *
```

Figure 5.0 JAXB Generated Classes

Defining the Service

The next step is to define the Service interface using the types we generated above. The Service interface is defined below and is very simple indeed.

```
package com.blog.samples.services;
import com.blog.samples.webservices.Account;
/**
 * The Interface AccountService.
 */
public interface AccountService
{
     /**
      * Gets the account details.
      *
      * @param accountNumber the account number
      * @return the account details
      */
     public Account getAccountDetails(String accountNumber);
}
```

Now we'll provide a really simple implementation of this interface. As you can see our service implementation returns some hard coded values. Obviously a real service implementation would do something more meaningful.

```
package com.blog.samples.services;
import org.springframework.stereotype.Service;
import com.blog.samples.webservices.Account;
import com.blog.samples.webservices.EnumAccountStatus;
/**
 * The Class AccountService.
 */
@Service
public class AccountServiceImpl implements AccountService
{
     /**
      * Gets the account details.
      *
      * @param accountNumber the account number
      * @return the account details
      */
     public Account getAccountDetails(String accountNumber)
     {
          /* hard coded account data - in reality this data would be retrieved
           * from a database or back end system of some sort */
          Account account = new Account();
          account.setAccountNumber("12345");
          account.setAccountStatus(EnumAccountStatus.ACTIVE);
          account.setAccountName("Joe Bloggs");
          account.setAccountBalance(3400);
          return account;
     }
}
```

Creating the Service Endpoint

A service endpoint is the component that deals with processing web service requests and responses. In the background a Spring Servlet intercepts incoming SOAP requests for a defined URL and routes them to an endpoint for processing. Below we're going to define that endpoint.

```
package com.blog.samples.services.endpoints;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;
import com.blog.samples.services.AccountService;
import com.blog.samples.webservices.Account;
import com.blog.samples.webservices.accountservice.AccountDetailsRequest;
import com.blog.samples.webservices.accountservice.AccountDetailsResponse;
/**
 * The Class AccountService.
 */
@Endpoint
public class AccountServiceEndpoint
{
     private static final String TARGET_NAMESPACE = "http://com/blog/samples/webservices/ac
     @Autowired
     private AccountService accountService_i;
     /**
      * Gets the account details.
      *
      * @param accountNumber the account number
      * @return the account details
      */
     @PayloadRoot(localPart = "AccountDetailsRequest", namespace = TARGET_NAMESPACE)
     public @ResponsePayload AccountDetailsResponse getAccountDetails(@RequestPayload Accou
     {
          AccountDetailsResponse response = new AccountDetailsResponse();
          /* call Spring injected service implementation to retrieve account data */
          Account account = accountService_i.getAccountDetails(request.getAccountNumber());
          response.setAccountDetails(account);
          return response;
34:        }
35:        public void setAccountService(AccountService accountService_p)
36:        {
37:             this.accountService_i = accountService_p;
38:        }
39:  }
```

Our sample application makes sue of Springs Web Services annotation support. The above class uses a number of these annotations, each of which is explained below.

Line 14 - @Enpoint is a specialised version of the standard Spring @Component annotation and allows this class to get picked up and registered by Springs component scanning.
Lines 18 & 19 - Our simple service implementation is Spring injected so that it can be used by our web service endpoint.
Line 17 - this is the namespace we defined in our XSD type definitions earlier. We use this in the endpoint class for mapping request to specific methods for processing.
Line 26 - @PayloadRoot indicates that this method will process service requests with the XML root element matching that defined by the localPart attribute. In the example above our method will process incoming requests of type AccountDetailsRequest with namespace http://com/blog/samples/webservices/accountservice. Remember that we defined this XSD type and namespace earlier.
Line 27 - @ResponsePayload indicates the type to be returned in the SOAP response. In this example the AccountDetailsResponse object will be converted to XML and returned to the client application as a SOAP response. @RequestPayload AccountDetails tells Spring to convert incoming requests of type AccountDetails, from XML to Java and the pass that object as a parameter to this endpoint method.

Spring Configuration

Next we'll write our Spring configuration to bring everything together. The Spring configuration is defined as follows.

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <beans xmlns="http://www.springframework.org/schema/beans"
3:            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4:             xmlns:context="http://www.springframework.org/schema/context"
5:             xmlns:sws="http://www.springframework.org/schema/web-services"
6:             xsi:schemaLocation="http://www.springframework.org/schema/beans
7:                                      http://www.springframework.org/schema/beans/spring-beans-
8:                                      http://www.springframework.org/schema/web-services
9:                                      http://www.springframework.org/schema/web-services/web-se
10:                                     http://www.springframework.org/schema/context
11:                                     http://www.springframework.org/schema/context/spring-con
12:       <context:component-scan base-package="com.blog.samples.services" />
13:       <sws:annotation-driven />
14:       <!--
15:            Our test service bean
16:       -->
17:       <bean id="AccountDetailsService" class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl
18:      <property name="schemaCollection">
19:        <bean class="org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection">
20:          <property name="inline" value="true" />
21:          <property name="xsds">
22:            <list>
23:              <value>schemas/AccountDetailsServiceOperations.xsd</value>
24:            </list>
25:          </property>
26:        </bean>
27:      </property>
28:      <property name="portTypeName" value="AccountDetailsService"/>
29:      <property name="serviceName" value="AccountDetailsServices" />
30:      <property name="locationUri" value="/endpoints"/>
31:    </bean>
32:  </beans>
```

Line 12 - Component scanning scans the defined package (com.blog.sample.services) for Spring managed components to load into the bean factory.
Line 13 - Enables Spring Web Services annotation support so that annotations like @PayloadRoot can

be used to configure the service endpoint.
Line 17 to 31 - Use of DefaultWsdl11Definition enables automated WSDL generation. Spring uses the schema definitions listed in the schemaCollection property, as well as the portType, serviceName and locationUri to generate a WSDL file the first time it is requested. Although this is a powerful feature it should be used with caution in production as the WSDL generation process can be quite slow. An approach I've used in the past is to copy the generated WSDL from your browser to your project and expose it using Springs static WSDL support with <static-wsdl>.

Web.xml

Now for the final bit of configuration before we test out our service. Web.xml is defined as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns="http://java.sun.com/xml/ns/javaee"
           xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
           id="WebApp_ID"
           version="2.5">
     <context-param>
          <param-name>contextConfigLocation</param-name>
           <param-value>
                /WEB-INF/config/spring-config.xml
           </param-value>
      </context-param>
      <listener>
           <listener-class>org.springframework.web.context.ContextLoaderListener</listener-c
      </listener>
      <servlet>
           <servlet-name>webservices</servlet-name>
           <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</se
           <init-param>
                <param-name>transformWsdlLocations</param-name>
                <param-value>true</param-value>
```

```
23:            </init-param>
24:            <init-param>
25:                 <param-name>contextConfigLocation</param-name>
26:                 <param-value></param-value>
27:            </init-param>
28:            <load-on-startup>1</load-on-startup>
29:       </servlet>
30:       <servlet-mapping>
31:            <servlet-name>webservices</servlet-name>
32:            <url-pattern>*.wsdl</url-pattern>
33:       </servlet-mapping>
34:       <servlet-mapping>
35:            <servlet-name>webservices</servlet-name>
36:            <url-pattern>/endpoints/*</url-pattern>
37:       </servlet-mapping>
38:  </web-app>
```

Line 8 to 13 - Path for Spring configuration to be loaded on application start-up.
Line 14 to 16 - Loads the Spring application context using the configuration file defined above
Line 18 to 19 - Spring Web Service Servlet that intercepts incoming HTTP requests.
Line 21 to 22 - Ensures WSDL is context aware. Transforms SOAP address so that it isn't hard coded to localhost:8080.Address updates depending on the application context and port that the application is deployed at.
Line 25 to 26 - ContextConfigLocation set with an empty parameter means that Spring won't try to load the default webservices-servlet.xml configuration.
Line 47 to 55 - Configures the URLs that our newly configured Web Services Servlet will handle.

Deploying the Service

We're now ready to deploy our application - I use Tomcat but feel free to use any Servlet container. Once the application is deployed, just browse to http://localhost:8080/spring-webservices-sample/endpoints/AccountDetailsService.wsdl and the application should generate and display the following WSDL.

```
1:  <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:sch0="http://com/blog/
```

```
        <wsdl:types>
             <xsd:schema xmlns="http://com/blog/samples/webservices/accountservice" xmlns:accou
                  <xsd:import namespace="http://webservices.samples.blog.com"/>
                  <xsd:element name="AccountDetailsRequest">
                       <xsd:complexType>
                            <xsd:sequence>
                                 <xsd:element name="accountNumber" type="xsd:string"/>
                            </xsd:sequence>
                        </xsd:complexType>
                   </xsd:element>
                   <xsd:element name="AccountDetailsResponse">
                        <xsd:complexType>
                             <xsd:sequence>
                                  <xsd:element name="AccountDetails" type="account:Account"/>
                             </xsd:sequence>
                        </xsd:complexType>
                   </xsd:element>
              </xsd:schema>
              <xs:schema xmlns="http://webservices.samples.blog.com" xmlns:xs="http://www.w3.or
                   <xs:element name="Account" type="Account"/>
                   <xs:complexType name="Account">
                        <xs:sequence>
                             <xs:element name="AccountNumber" type="xs:string"/>
                             <xs:element name="AccountName" type="xs:string"/>
                             <xs:element name="AccountBalance" type="xs:double"/>
                             <xs:element name="AccountStatus" type="EnumAccountStatus"/>
                        </xs:sequence>
                   </xs:complexType>
                   <xs:simpleType name="EnumAccountStatus">
                        <xs:restriction base="xs:string">
                             <xs:enumeration value="Active"/>
                             <xs:enumeration value="Inactive"/>
                        </xs:restriction>
```

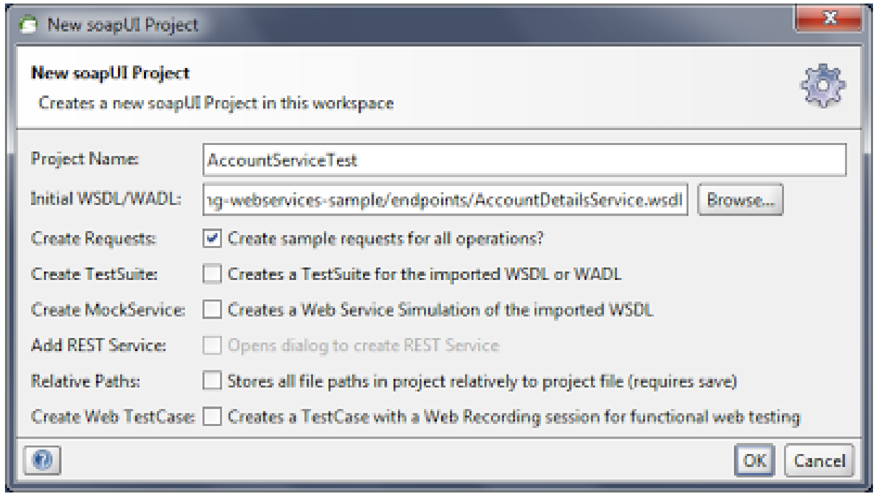
```
                </xs:simpleType>
            </xs:schema>
        </wsdl:types>
        <wsdl:message name="AccountDetailsResponse">
            <wsdl:part element="tns:AccountDetailsResponse" name="AccountDetailsResponse"/>
        </wsdl:message>
        <wsdl:message name="AccountDetailsRequest">
            <wsdl:part element="tns:AccountDetailsRequest" name="AccountDetailsRequest"/>
        </wsdl:message>
        <wsdl:portType name="AccountDetailsService">
            <wsdl:operation name="AccountDetails">
                <wsdl:input message="tns:AccountDetailsRequest" name="AccountDetailsRequest"
                <wsdl:output message="tns:AccountDetailsResponse" name="AccountDetailsRespon
            </wsdl:operation>
        </wsdl:portType>
        <wsdl:binding name="AccountDetailsServiceSoap11" type="tns:AccountDetailsService">
            <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="AccountDetails">
                <soap:operation soapAction=""/>
                <wsdl:input name="AccountDetailsRequest">
                    <soap:body use="literal"/>
                </wsdl:input>
                <wsdl:output name="AccountDetailsResponse">
                    <soap:body use="literal"/>
                </wsdl:output>
            </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="AccountDetailsServices">
            <wsdl:port binding="tns:AccountDetailsServiceSoap11" name="AccountDetailsServiceS
                <soap:address location="http://localhost:8080/spring-webservices-sample/endp
            </wsdl:port>
        </wsdl:service>
    </wsdl:definitions>
```

Testing the Service

The simplest way to test a SOAP service is using SoapUI. For anyone who hasn't used it before, SoapUI is an open source functional testing tool for testing SOAP web services. It saves us having to write a web service client and means that in just a few clicks we can have a test harness in place to test our service.

To test our serviced using SoapUI follow the steps below.

- Download and install SoapUI from here.
- Go to File->New SoapUI Project
- For project name enter AccountServiceTest
- For initial WSDL enter the path to your deployed WSDL - http://localhost:8080/spring-webservices-sample/endpoints/AccountDetailsService.wsdl



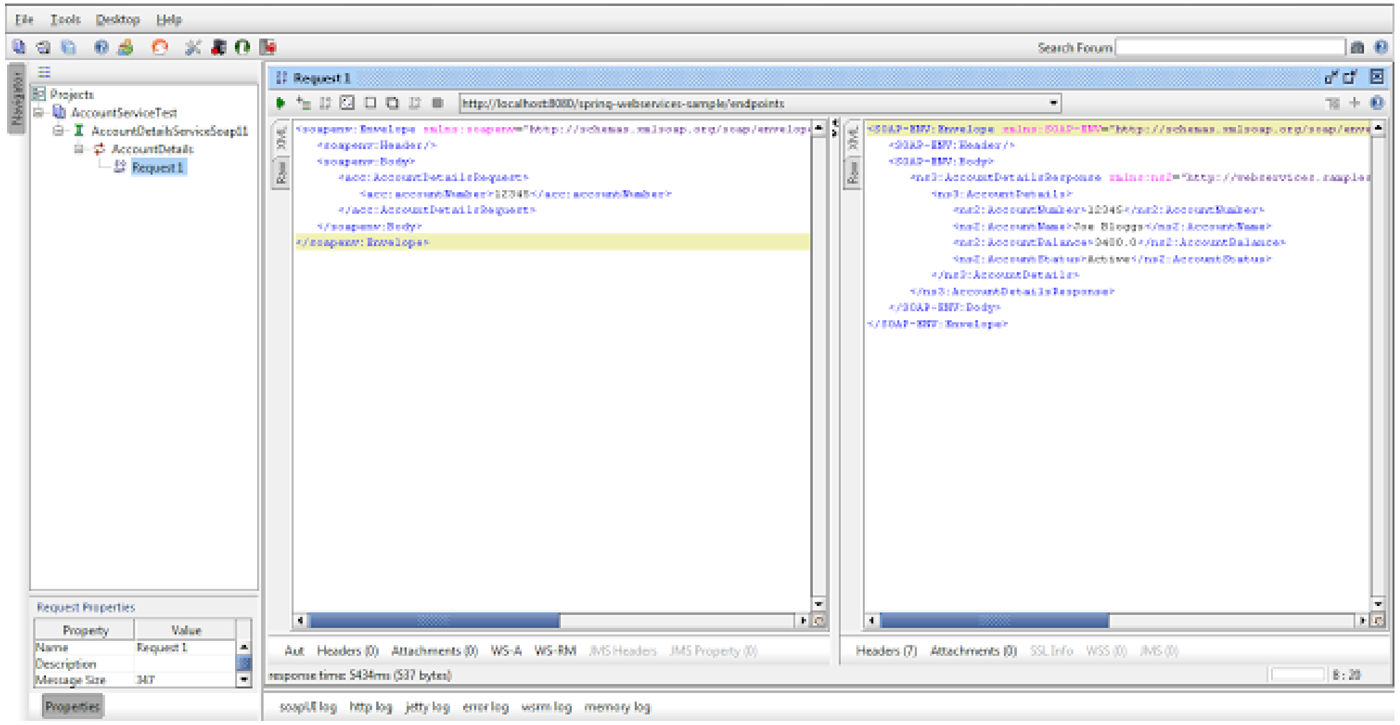
New soapUI Project

New soapUI Project
Creates a new soapUI Project in this workspace

Project Name:	AccountServiceTest
Initial WSDL/WADL:	ng-webservices-sample/endpoints/AccountDetailsService.wsdl
Create Requests:	[✓] Create sample requests for all operations?
Create TestSuite:	Creates a TestSuite for the imported WSDL or WADL
Create MockService:	Creates a Web Service Simulation of the imported WSDL
Add REST Service:	Opens dialog to create REST Service
Relative Paths:	Stores all file paths in project relatively to project file (requires save)
Create Web TestCase:	Creates a TestCase with a Web Recording session for functional web testing

Figure 6.0 SoapUI Test Project

- SoapUI will parse the exposed WSDL (make sure your application is deployed and the WSDL is exposed!) and use it to build a sample SOAP request.
- When the new project opens click AccountServiceTest -> AccountDetails -> request and you'll see a SOAP request for the AccountDetails service in the left hand pane. Set the account number and press the green arrow in the top left hand corner to call the service.
- If the request is successful you should see a SOAP response containing the requested account data in the right hand pane. See figure 7.0 below



File Tools Desktop Help
Search Forum
Projects
AccountServiceTest
AccountDetailsServiceSoap11
AccountDetails
Request 1
Request Properties
Property Value
Name Request 1
Description
Message Size 347
Properties
Aut Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (0)
Headers (7) Attachments (0) SSL Info WSS (0) JMS (0)
soapUI log http log jetty log error log wsrm log memory log

http://localhost:8080/spring-webservices-sample/endpoints

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelop
   <soapenv:Header/>
   <soapenv:Body>
      <acc:AccountDetailsRequest>
         <acc:accountNumber>12345</acc:accountNumber>
      </acc:AccountDetailsRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/enve
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns3:AccountDetailsResponse xmlns:ns2="http://webservices.samples
         <ns3:AccountDetails>
            <ns2:AccountNumber>12345</ns2:AccountNumber>
            <ns2:AccountName>Joe Bloggs</ns2:AccountName>
            <ns2:AccountBalance>2400.0</ns2:AccountBalance>
            <ns2:AccountStatus>Active</ns2:AccountStatus>
         </ns3:AccountDetails>
      </ns3:AccountDetailsResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

response time: 5434ms (537 bytes)   8 : 20

Figure 7.0 SoapUI AccountDetaills Service Test

Summary

The sample code in this post took you through the steps required to build, deploy and test a contract first web service using the Spring framework. Don't forget that you can download the full source code for this tutorial and play around with it. If you found this tutorial useful then feel free to share it with others or leave a comment below. The full source code can be found on GitHub at https://github.com/briansjavablog/spring-webservices-tutorial.

Posted by Brian at 10:16

+10 Recommend this on Google

Labels: Contract First Web Service, Java Web Services, SOAP Services, Spring Web Services, web services

80 comments:

Anshul 4 February 2013 at 03:00

the best tutorial ever worked , thank you sir

it really helped ...all other blogs sucks for thier tutorial even spring official framework developers especially arjen !! :)

Reply

Replies

Brian 4 February 2013 at 03:34

Thanks Anshul, glad you liked it.

Reply

Pavan Kumar 8 February 2013 at 00:21

This comment has been removed by the author.

Reply

Pavan Kumar 8 February 2013 at 00:32

HI, Brian,
It was very helpful for me in easy understanding the Spring WS.
Even i had gone threw many blogs. I didn't got the detailed explanation like this..
Keep on blogging the new technologies ......
Thanks once again..

Reply

Ammar 8 February 2013 at 17:35

Thank you so much.
Great Tutorial.

Reply

nityanandasahoo 9 February 2013 at 11:54

It is nice but you have not developed the client........i need the client also.please post it.

Reply

Pankaj Verma 3 March 2013 at 16:40

A very nice and comprehensive tutorial. Admire the beautiful presentation style and completeness! Great job and, of course, Many Thanks for taking the pains to put this up.

Reply

Skubanek 7 March 2013 at 01:17

I spend a day searching for something like this - thanks a lot for the detailed explanations provided.

Reply

Venkatraman 14 March 2013 at 10:03

Awesome tutorial:) Thank you so much. If possible can you develop a client too for this?

Reply

Replies

Brian 14 March 2013 at 10:31

I've posted an Axis2 web service client tutorial here

Reply

casimirrex antony 21 March 2013 at 23:49

Hi This source code does not available in following url(https://github.com/briansjavablog/spring-webservices-tutorial)

Pls ensure and give and actual soure code url

-Rex

Reply

Replies

Brian 22 March 2013 at 14:37

I just checked the link and it works fine.

Reply

Ernesto Diaz 22 March 2013 at 04:36

This is an outstanding job! First example (I found) that really works, I hope someone will put it on the official Spring site. Thanks!!

Reply

Vijay Prakash Vyas 23 March 2013 at 16:13

Gr8 job Brian, Using above sample I converted my existing WS project from Cxf to Spring based.

Reply

shivam verma 26 March 2013 at 01:57

Hey, i have read this one . Its really great.

One thing, if you could just help out!!!

How do we access , these web services through our java classes.

I mean simply from any java class/servlet , how do i send a particular request, and immediately recieve a response as well. !!!!

Reply

Brian 26 March 2013 at 05:29

Sounds like you're looking for a web service client - you can see an example here using Axis2.

Reply

shivam verma 27 March 2013 at 02:35

hello brian , i am thinking of accessing the web service through url. i mean both applications are different. And i want to access it through sending a request.

Moreover, i would also like to have a word from you. On just letting me know, if Spring web service would be good for usage or packages like these -- apache axis, raw jax , ...Etc. you can specify too.

That would be a appreciated guided help. !!!

Reply

shivam verma 27 March 2013 at 02:43

ok , i think i sort of overlooked your reply , your sample.... Well, i see it is sort of a client... But then that is almost very complex thing....

...

another thing, while making the web service , as i was.. i had in my head. That it would be something standalone. ANd accessable directly via url's ... Well, one more thing, the way you are depicting -- >

is it the standard protocol/ or are there any other methods as well...

Guided Help would be highly appreciated ...

Reply

test 31 March 2013 at 16:20

Very good tutorial

Reply

Raúl Garcia 1 April 2013 at 12:43

This comment has been removed by the author.

Reply

neel4Tech 5 April 2013 at 03:48

Very nice explanation, thanks a lot :)

Reply

Binu 13 April 2013 at 02:42

Brian,

Very good explanation. I ran this in Ubuntu 11.10, Spring.
Source Tool Suite. Worked without any major problem. Somehow it is not able to detect the spring-context.xml at /WEB-INF/conf location.

I just moved that into /WEB-INF location. It worked then.

Thanks a lot

Reply

rahul urama 27 April 2013 at 11:11

great..awesome tutorial

Reply

ddd 27 April 2013 at 15:34

Hi Brian,

i got the following error/exception when i tried to deploy the project to Tomcat:Any idea how to solve it..

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'AccountDetailsService' defined in ServletContext resource [/WEB-INF/config/spring-config.xml]: Initialization of bean failed; nested exception is java.lang.NoClassDefFoundError: org/apache/ws/commons/schema/resolver/URIResolver
at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:527)
at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:456)
at
org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:294)
at
org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:225)
at
org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:291)
at
org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.jav

```
a:197)
at
org.springframework.beans.factory.support.DefaultListableBeanFactory.getBeansOfType(DefaultLis
tableBeanFactory.java:400)
at
org.springframework.context.support.AbstractApplicationContext.getBeansOfType(AbstractApplicati
onContext.java:1164)
at
org.springframework.beans.factory.BeanFactoryUtils.beansOfTypeIncludingAncestors(BeanFactory
Utils.java:275)
at
org.springframework.beans.factory.BeanFactoryUtils.beansOfTypeIncludingAncestors(BeanFactory
Utils.java:279)
at
org.springframework.ws.transport.http.MessageDispatcherServlet.initWsdlDefinitions(MessageDispa
tcherServlet.java:385)
at
org.springframework.ws.transport.http.MessageDispatcherServlet.initFrameworkServlet(MessageDis
patcherServlet.java:231)
at org.springframework.web.servlet.FrameworkServlet.initServletBean(FrameworkServlet.java:307)
at org.springframework.web.servlet.HttpServletBean.init(HttpServletBean.java:127)
at javax.servlet.GenericServlet.init(GenericServlet.java:160)
at org.apache.catalina.core.StandardWrapper.initServlet(StandardWrapper.java:1280)
at org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1193)
at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:1088)
at org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:5123)
at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5407)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)
at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1559)
at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1549)
at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
at java.util.concurrent.FutureTask.run(FutureTask.java:166)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)
at java.lang.Thread.run(Thread.java:722)
Caused by: java.lang.NoClassDefFoundError:
org/apache/ws/commons/schema/resolver/URIResolver
```

Reply

Brian 29 April 2013 at 00:26

It looks like Tomcat can't find org.apache.ws.commons.schema.resolver.URIResolver.class. This class should be included in XmlSchema version 1.4.3. (set up as a POM dependency with artifact Id XmlSchema).

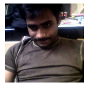
Reply

Replies

ddd 6 May 2013 at 07:26

You're right. somehow i was missing the XmlSchema dependency from my pom. my fualt!...thanks for the reply!

Reply



Chandu Chinthala 1 May 2013 at 20:08

Hi Brain,

It is a nice tutorial for creating a web-service. I am following the exact steps mentioned, but after generating Java classes that are related to XSD/schema, somehow Java files are unable to any of the

javax.xml.bind.* classes like XmlType, XmlAccessType.. etc. I tried adding below dependancies to pom.xml

javax.xml
jaxb-api
2.1

com.sun.xml.bind
jaxb-impl
2.2.7

Then the compilation errors are gone, but while accessing the service I am facing like " org.springframework.beans.BeanInstantiationException: Could not instantiate bean class [org.springframework.ws.server.endpoint.adapter.method.SourcePayloadMethodProcessor]: Constructor threw exception; nested exception is java.lang.ClassCastException: com.sun.xml.stream.ZephyrParserFactory "

You have any idea?.. Do we really need the dependencies I have added or the dependacies you have in your example are enough?..

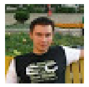
thanks for your help

Reply

Brian Heisler 24 May 2013 at 12:41

Excellent work, well done - thanks!

Reply



Georgian Micsa 4 June 2013 at 14:36

Hi,

I`m using apache-tomcat-6.0.37, jdk1.6.0_45 and SOAPUI 4.5.2. Although I can access the WSDL in the browser, importing it in SOAPUI fails with
Error loading [http://localhost:8080/spring-webservices-sample/endpoints/AccountDetailsService.wsdl]: java.lang.Exception: Failed to load url; http://localhost:8080/spring-webservices-sample/endpoints/AccountDetailsService.wsdl but no exception in the logs.
I also tried to save the WSDL in a file and import it in SOAPUI but when I call the method it always gives socket timeout exception.

Do you have any idea about this weird situation?

Thanks,
Georgian

Reply

▾ Replies

Georgian Micsa 5 June 2013 at 13:03

It seems to be only a SOAPUI problem, I could call the web service from a Java client.

Reply

Niha 11 June 2013 at 13:46

This comment has been removed by the author.

Reply

Dario 14 June 2013 at 08:42

Great work. Clear explanation.

Thanks

Best

Reply

Unknown 8 July 2013 at 10:05

Would you please explain what type of Project is created and which eclipse. I am trying to use Eclipse 3.5.2. I can see Dynamic/Static/Web service project. I am new to Web Services and all, but know Java. I am having some trouble to select which project to do.

Reply

▾ Replies

Brian 8 July 2013 at 23:50

Instead of creating a new project using an Eclipse template you'll need to import the Maven project. GO to File->Import->Maven->Existing Maven Projects and point it at the POM.

Reply

Bala 11 July 2013 at 12:35

This comment has been removed by the author.

Reply

▾ Replies

Bala 11 July 2013 at 12:37

Excellent work with crystal clear explanations. Thanks for this wonderful work Brian!!.

Reply

Ivan Evseev 19 July 2013 at 07:48

Great article!

Reply

naryan 2 August 2013 at 21:39

How do you display/log the complete raw SOAP request/response from webservice client that is generated using axis2.

Reply

George Trandafir 19 August 2013 at 08:53

Hello!
I am using jaxb2 and facing the following error:

```
org.xml.sax.SAXParseException; systemId: file:/C:/Users/gtrandafir/workspace_kepler/PATServer/src/main/webapp/schemas/HRISDetailsServiceOperations.xsd; lineNumber: 19; columnNumber: 45; src-resolve: Cannot resolve the name 'hris:HRIS' to a(n) 'type definition' component.
at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.createSAXParseException(ErrorHandlerWrapper.java:198)
at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.error(ErrorHandlerWrapper.java:134)
at com.sun.org.apache.xerces.internal.impl.XMLErrorReporter.reportError(XMLErrorReporter.java:437)
at com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDHandler.reportSchemaErr(XSDHandler.java:4124)
at com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDHandler.reportSchemaError(XSDHandler.java:4107)
at com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDHandler.getGlobalDecl(XSDHandler.java:1730)
at com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDElementTraverser.traverseNamedElement(XSDElementTraverser.java:405)
at com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDElementTraverser.traverseLocal(XSDElementTraverser.java:194)
at com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDHandler.traverseLocalElements(XSDHandler.java:3580)
```

```
at
com.sun.org.apache.xerces.internal.impl.xs.traversers.XSDHandler.parseSchema(XSDHandler.java:
622)
at
com.sun.org.apache.xerces.internal.impl.xs.XMLSchemaLoader.loadSchema(XMLSchemaLoader.jav
a:588)
at
com.sun.org.apache.xerces.internal.impl.xs.XMLSchemaLoader.loadGrammar(XMLSchemaLoader.j
ava:555)
at
com.sun.org.apache.xerces.internal.impl.xs.XMLSchemaLoader.loadGrammar(XMLSchemaLoader.j
ava:521)
at
com.sun.org.apache.xerces.internal.jaxp.validation.XMLSchemaFactory.newSchema(XMLSchemaFa
ctory.java:240)
at
com.sun.tools.xjc.reader.xmlschema.parser.SchemaConstraintChecker.check(SchemaConstraintCh
ecker.java:101)
at com.sun.tools.xjc.ModelLoader.loadXMLSchema(ModelLoader.java:357)
at com.sun.tools.xjc.ModelLoader.load(ModelLoader.java:167)
at com.sun.tools.xjc.ModelLoader.load(ModelLoader.java:113)
at com.sun.tools.xjc.Driver.run(Driver.java:313)
at
```

Any ideas, please?
PS: i am using maven from CLI.

Reply

Replies

Brian 19 August 2013 at 23:09

Hi George.
The first step is to validate your XSD document and make sure there are no issues. If you open the document in XMLSpy you can validate it by pressing F8. If there are issues, XMLSpy will highlight the point of failure which should make the problem easy to resolve.

thanks
Brian

Reply

M Zaky 22 August 2013 at 16:20

I could not build with maven:
org.apache.maven.lifecycle.LifecycleExecutionException: Failed to execute goal org.codehaus.mojo:jaxb2-maven-plugin:1.4:xjc (default-cli) on project spring-webservices-sample: Could not process schema files in directory
Notice that the schemas are located in : src/main/webapp/schemas
Any idea?
Thanks,
mzaky

Reply

Replies

M Zaky 24 August 2013 at 04:47

This comment has been removed by the author.

M Zaky 24 August 2013 at 04:59

Hi all,

I resolved it, when I changed the version of the plugin to 1.5 and I put the schemas files in the right folder (as defined in the documentation of the plugin, http://mojo.codehaus.org/jaxb2-maven-plugin/xjc-mojo.html):

I hope it will help!

Kind regards

mzaky

Reply

npeducations 30 September 2013 at 22:01

thaks for the valuable information which made me more helpful in my project work.

Reply

Diego Herrera 20 October 2013 at 21:54

I have the error I ws.server.EndpointNotFound. some solution?

```
DEBUG: [oct-21 01:48:18,355] transport.http.WsdlDefinitionHandlerAdapter - Transforming
[/endpoints] to [http://localhost:8080/SpringWebService01/endpoints]
DEBUG: [oct-21 01:48:18,361] transport.http.MessageDispatcherServlet - Successfully completed
request
DEBUG: [oct-21 01:48:31,820] transport.http.WebServiceMessageReceiverHandlerAdapter -
Accepting incoming [org.springframework.ws.transport.http.HttpServletConnection@e3d4e8] at
[http://localhost:8080/SpringWebService01/endpoints]
DEBUG: [oct-21 01:48:31,862] server.MessageTracing.received - Received request
[SaajSoapMessage {http://com/company/webservices/accountservice}AccountDetailsRequest]
DEBUG: [oct-21 01:48:31,893] endpoint.mapping.PayloadRootAnnotationMethodEndpointMapping -
Looking up endpoint for [{http://com/company/webservices/accountservice}AccountDetailsRequest]
DEBUG: [oct-21 01:48:31,893] soap.server.SoapMessageDispatcher - Endpoint mapping
[org.springframework.ws.server.endpoint.mapping.PayloadRootAnnotationMethodEndpointMapping
@1a954f8] has no mapping for request
DEBUG: [oct-21 01:48:31,894] endpoint.mapping.SoapActionAnnotationMethodEndpointMapping -
Looking up endpoint for []
DEBUG: [oct-21 01:48:31,894] soap.server.SoapMessageDispatcher - Endpoint mapping
[org.springframework.ws.soap.server.endpoint.mapping.SoapActionAnnotationMethodEndpointMap
ping@9770b] has no mapping for request
WARN : [oct-21 01:48:31,894] ws.server.EndpointNotFound - No endpoint mapping found for
[SaajSoapMessage {http://com/company/webservices/accountservice}AccountDetailsRequest]
```

```
DEBUG: [oct-21 01:48:31,895] transport.http.MessageDispatcherServlet - Successfully completed request
```

Reply

▾ Replies

usefulpiecesofcode 5 November 2013 at 13:14

Diego, I also had this problem. You need to make sure you have carefully typed the spring-config.xml as well as having matching variable names / paths in AccountServicesEndpoints.java. The problem is exactly as the logs state, the endpoint which you have specified cannot be found.

Reply

Michael McCabe 30 October 2013 at 06:24

Brian, this is the best tutorial I've ever seen on spring java WS. Keep up the good work!

Reply

usefulpiecesofcode 5 November 2013 at 13:10

Hey Brian. Thanks a million for the tutorial. Michaels statement (above) rings true for me too. I have followed your approach whilst building my own application. I am returning hardcoded data from the Account class.

As an additional step, I am trying to add Hibernate to the project. The problem is, when I add the relevant annotations to the JAXB generated Account class, they get overwritten. Do you know what is the correct way to go about this?

Thanks

Reply

Rat Net 23 November 2013 at 02:59

Hello All, I've no compile time error in my eclipse project but at runtime the application is not working and presented the following error logs:

```
INFO: Loading XML bean definitions from ServletContext resource [/WEB-INF/config/spring-config.xml]
????? ??, ???? ?:??:?? ??????? org.springframework.web.context.ContextLoader initWebApplicationContext
SEVERE: Context initialization failed
org.springframework.beans.factory.BeanDefinitionStoreException: Unexpected exception parsing XML document from ServletContext resource [/WEB-INF/config/spring-config.xml]; nested exception is java.lang.NoClassDefFoundError: org/springframework/xml/transform/TransformerObjectSupport
```

Reply

Gomathi T 23 November 2013 at 21:20

Thanks for the tutorial. Its very clear for new learners of Spring WebServices Configuration. Thanks for the excellent one...

Reply

raj 3 December 2013 at 06:22

Hi,

This tutorial is looking good but actually I am new in Spring and I have to create Spring web service. So can anybody help me to write from starting like how to create the spring project and how JAXB can be used to generate the classes from XSD. If anybody have details document please provide me .
Thanks.

Reply

Swamy 4 December 2013 at 03:18

thanks .As i am working on spring web services.It is useful for me.

Reply

Divya Besant 25 December 2013 at 22:05

Nice Blog contact for more info

Reply

ROHIT JAIN 4 January 2014 at 04:31

Awesome Blog.....thanks!!!

Reply

blue dream 10 February 2014 at 07:28

Perfect, Thanks!! This should be replaced by this: http://docs.spring.io/spring-ws/site/reference/html/tutorial.html

Do you also have any Idea how to call this Web service from java script or Ajax or anything on a HTML???

Reply

Rogério Reis Batista 7 March 2014 at 12:35

Very, very good. I liked so much. Now I could understand how spring ws works. Thank you for this tutorial.

Reply

laffu 8 March 2014 at 13:10

very nice tutorial.

one thing I noticed though is that .. in soapUI, you pass any account number, it responds with the 12345 details.

Reply

Olive Arnold 3 April 2014 at 04:09

Graet post

php development

Reply

Rashmi Singh 19 May 2014 at 13:47

Hi Brian, Thanks for the useful post. I was able to get it to work. I had a question though. Can you provide some help on how to do the same using Gradle instead of Maven? Do you have a build.gradle corresponding to the pom.xml that would essentially help do the same build?

Reply

Akif Khan 6 June 2014 at 04:57

Hi Brian, This is really an awesome tutorial for beginners, could you please identify what are the possible jars required to run this project.

Reply

▾ Replies

Brian 6 June 2014 at 05:20

All required dependencies are defined in the POM.xml. When you run a Maven build these resources will be downloaded to your local Maven repository (if you don't already have them) and get added to your applications class path.

Reply

Dunith Dhanushka 8 June 2014 at 00:06

Well, this made my day. Very comprehensive tutorial indeed. Thanks for sharing!

Reply

Shubham Goel 11 June 2014 at 15:17

Now this was one awesome tutorial for a complete spring WS beginner !!

Reply

sluggo 19 June 2014 at 13:30

Took a bit to get the right XmlSchema jar downloaded (Use:

org.apache.ws.xmlschema
xmlschema-core
2.0.1
)
But otherwise worked GREAT!!

Reply

Santhosh Mamulla 25 June 2014 at 05:33

Hi Brian,
It is best tutorial for beginners to understand Spring WS
Thanks for this post.. :)

Reply

Anbukarasi J 29 June 2014 at 04:20

This comment has been removed by the author.

Reply

Sophia Right 18 July 2014 at 02:01

Great post. I Like to read your post. I gonna to bookmark your post. Thanks for sharing all links I am sure they are useful.

Reply

Hosting Raja 5 August 2014 at 00:54

This is an excellent blog because it has good volume of information, everything is described in the simplest manner and all information on this blog is genuine and real..best web hosting | web hosting sites

Reply

George Celvi 8 August 2014 at 17:19

I am trying to access the WebServiceContext within an operation and have ..
@Resource
public void setWebServiceContext(WebServiceContext c) {
wsCtxt = c;
}
added this code to the AccountServiceImpl class
It always returns null...
Is there a way to access the WebServiceContext?
Thanks
George

Reply

santosh raju 25 August 2014 at 03:02

hi brain...can u please re-Commit the whole project..many packages are missing after checking out the project...great tutorial..thanks

Reply

santosh raju 25 August 2014 at 04:07

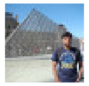
hi brian...can u please re-Commit the whole project..many packages are missing after checking out the project...great tutorial..thanks

Reply

Rima 13 September 2014 at 12:11

Thanks Brian for a such wonderful blog.It helped me lot.

Reply



Makky 15 September 2014 at 06:44

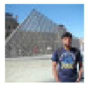
Its missing lots of things. You forgot to add wsdl4j and other deps.

Reply

bala10 15 September 2014 at 21:55

Hi Brain,
Can you please brief a little on adding SOAP Header to the above web service example

Reply



Makky 21 September 2014 at 01:58

This is complicated. This can be done with Apache CXF easily...

For full tutorial (WSDL, Server and Client) follow here

https://www.youtube.com/watch?v=EehEBC_LWzY

Reply

mary jane 13 October 2014 at 19:52

Hi Brian,

I am doing a project both web app and web services in it. I put web app files into a package: com.motelming.j2ee.webapp. I put web services files into another package: com.motelming.j2ee.services. Now I tried to get the .wsdl file and failed. Here is the error message:

http://localhost:8082/com.motelming.j2ee.webapp/endpoints/RoomDetailsService.wsdl

```
2014-10-13 22:42:51 ERROR MessageDispatcherServlet:492 - Context initialization failed
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'RoomDetailsService' defined in ServletContext resource [/WEB-INF/spring-config.xml]: Cannot create inner bean 'org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection#76c99fd9' of type [org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection] while setting bean property 'schemaCollection'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection#76c99fd9' defined in ServletContext resource [/WEB-INF/spring-config.xml]: Invocation of init method failed; nested exception is java.lang.NoSuchMethodError: org.apache.ws.commons.schema.XmlSchemaCollection.read(Lorg/xml/sax/InputSource;)Lorg/apache/ws/commons/schema/XmlSchema;


Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection#76c99fd9' defined in ServletContext resource [/WEB-INF/spring-config.xml]: Invocation of init method failed; nested exception is java.lang.NoSuchMethodError: org.apache.ws.commons.schema.XmlSchemaCollection.read(Lorg/xml/sax/InputSource;)Lorg/apache/ws/commons/schema/XmlSchema;

Caused by: java.lang.NoSuchMethodError: org.apache.ws.commons.schema.XmlSchemaCollection.read(Lorg/xml/sax/InputSource;)Lorg/apach
```

```
e/ws/commons/schema/XmlSchema;

---==--------------------
HTTP Status 404 - /com.motelming.j2ee.services/endpoints/RoomDetailsService.wsdl


type Status report

message /com.motelming.j2ee.services/endpoints/RoomDetailsService.wsdl

description The requested resource is not available.
```

Reply

Allan 31 October 2014 at 04:25

This comment has been removed by a blog administrator.

Reply

Amit 5 November 2014 at 09:49

This comment has been removed by the author.

Reply

Alecia Us 7 November 2014 at 07:37

Providing Bangladesh Garments Exporters, Bangladesh Garment Factory and Industry, Bangladesh Clothing Manufacturers and Bangladesh Textile Industry List.

Reply

Zoltan 25 November 2014 at 14:15

Hi Brian, great blog.
Can you explain, how to add basic authentication to the service getAccountDetails ? (username,

password)
Thanks.

Reply

Enter your comment...

Comment as: Select profile...

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)