JAX-WS Configuration

Please see the Configuration section to learn how to supply a configuration to CXF. The following sections include just JAX-WS specific configuration items.

Configuring an Endpoint

A JAX-WS Endpoint can be configured in XML in addition to using the JAX-WS APIs. Once you've created your server implementation, you simply need to provide the class name and an address. Here is a simple example:

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jaxws="http://cxf.apache.org/jaxws"
 xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

<jaxws:endpoint id="classImpl"
    implementor="org.apache.cxf.jaxws.service.Hello"
    endpointName="e:HelloEndpointCustomized"
    serviceName="s:HelloServiceCustomized"
    address="http://localhost:8080/test"
    xmlns:e="http://service.jaxws.cxf.apache.org/endpoint"
    xmlns:s="http://service.jaxws.cxf.apache.org/service"/>

</beans>
```

Be sure to include the JAX-WS `schemaLocation` attribute specified on the root beans element. This allows CXF to validate the file and is required. Also note the namespace declarations at the end of the <jaxws:endpoint/> tag--these are required because the combined "{namespace}localName" syntax is presently not supported for this tag's attribute values.

The `jaxws:endpoint` element (which appears to create an EndpointImpl under the covers) supports many additional attributes:

Name	Value
endpointName	The endpoint name this service is implementing, it maps to the wsdl:port@name. In the format of "ns:ENDPOINT_NAME" where ns is a namespace prefix valid at this scope.
publish	Whether the endpoint should be published now, or whether it will be published at a later point.
serviceName	The service name this service is implementing, it maps to the wsdl:service@name. In the format of "ns:SERVICE_NAME" where ns is a namespace prefix valid at this scope.
wsdlLocation	The location of the WSDL. Can be on the classpath, file system, or be hosted remotely.
bindingUri	The URI, or ID, of the message binding for the endpoint to use. For SOAP the binding URI(ID) is specified by the JAX-WS specification. For other message bindings the URI is the namespace of the WSDL extensions used to specify the binding.
address	The service publish address
bus	The bus name that will be used in the jaxws endpoint.
implementor	The implementor of jaxws endpoint. You can specify the implementor class name here, or just the ref bean name in the format of "#REF_BEAN_NAME"
implementorClass	The implementor class name, it is really useful when you specify the implementor with the ref bean which is wrapped by using Spring AOP

createdFromAPI	This indicates that the endpoint bean was already created using jaxws API's thus at runtime when parsing the bean spring can use these values rather than the default ones. It's important that when this is true, the "name" of the bean is set to the port name of the endpoint being created in the form "{http://service.target.namespace} PortName".
publishedEndpointUrl	The URL that is placed in the address element of the wsdl when the wsdl is retrieved. If not specified, the address listed above is used. This parameter allows setting the "public" URL that may not be the same as the URL the service is deployed on. (for example, the service is behind a proxy of some sort).

It also supports many child elements:

Name	Value
jaxws:executor	A Java executor which will be used for the service. This can be supplied using the Spring <bean class="MyExecutor"/> syntax.
jaxws:inInterceptors	The incoming interceptors for this endpoint. A list of <bean>s or <ref>s. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:inFaultInterceptors	The incoming fault interceptors for this endpoint. A list of <bean>s or <ref>s. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:outInterceptors	The outgoing interceptors for this endpoint. A list of <bean>s or <ref>s. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:outFaultInterceptors	The outgoing fault interceptors for this endpoint. A list of <bean>s or <ref>s. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:handlers	The JAX-WS handlers for this endpoint. A list of <bean>s or <ref>s. Each should implement javax.xml.ws.handler.Handler or javax.xml.ws.handler.soap.SOAPHandler (Note that `@HandlerChain` annotations on the service bean appear to be ignored)
jaxws:properties	A properties map which should be supplied to the JAX-WS endpoint. See below.
jaxws:dataBinding	Which DataBinding to use in the endpoint. This can be supplied using the Spring <bean class="MyDataBinding"/> syntax.
jaxws:binding	You can specify the BindingFactory for this endpoint to use. This can be supplied using the Spring <bean class="MyBindingFactory"/> syntax.
jaxws:features	The features that hold the interceptors for this endpoint. A list of <bean>s or <ref>s
jaxws:invoker	The invoker which will be supplied to this endpoint. This can be supplied using the Spring <bean class="MyInvoker"/> syntax.
jaxws:schemaLocations	The schema locations for endpoint to use. A list of <schemaLocation>s
jaxws:serviceFactory	The service factory for this endpoint to use. This can be supplied using the Spring <bean class="MyServiceFactory"/> syntax

Here is a more advanced example which shows how to provide interceptors and properties:

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jaxws="http://cxf.apache.org/jaxws"
        xmlns:soap="http://cxf.apache.org/bindings/soap"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
           http://cxf.apache.org/bindings/soap
           http://cxf.apache.org/schemas/configuration/soap.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-extension-soap.xml"/>

  <jaxws:endpoint
    id="helloWorld"
    implementor="demo.spring.HelloWorldImpl"
    address="http://localhost/HelloWorld">
    <jaxws:inInterceptors>
      <bean class="com.acme.SomeInterceptor"/>
      <ref bean="anotherInterceptor"/>
    </jaxws:inInterceptors>
    <jaxws:properties>
      <entry key="mtom-enabled" value="true"/>
    </jaxws:properties>
  </jaxws:endpoint>

  <bean id="anotherInterceptor" class="com.acme.SomeInterceptor"/>

  <jaxws:endpoint id="simpleWithBinding"
    implementor="#greeter"
        address="http://localhost:8080/simpleWithAddress">
   <jaxws:binding>
      <soap:soapBinding mtomEnabled="true" version="1.2"/>
   </jaxws:binding>
  </jaxws:endpoint>

  <jaxws:endpoint id="inlineInvoker"
    address="http://localhost:8080/simpleWithAddress">
    <jaxws:implementor>
      <bean class="org.apache.hello_world_soap_http.GreeterImpl"/>
    </jaxws:implementor>
    <jaxws:invoker>
      <bean class="org.apache.cxf.jaxws.spring.NullInvoker"/>
    </jaxws:invoker>
  </jaxws:endpoint>

</beans>
```

If you are a Spring user, you'll notice that the `jaxws:properties` element follows the Spring Map syntax.

Configuring a Spring Client (Option 1)

This technique lets you add a Web Services client to your Spring application. You can inject it into other Spring beans, or manually

retrieve it from the Spring context for use by non-Spring-aware client code.

The easiest way to add a Web Services client to a Spring context is to use the `<jaxws:client>` element (similar to the `<jaxws:endpoint>` element used for the server side). Here's a simple example:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       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.xsd
          http://cxf.apache.org/jaxws
          http://cxf.apache.org/schemas/jaxws.xsd">

    <jaxws:client id="helloClient"
                  serviceClass="demo.spring.HelloWorld"
                  address="http://localhost:9002/HelloWorld" />
</beans>
```

The attributes available on `<jaxws:client>` include:

Name	Type	Description
id	String	A unique identified for the client, which is how other beans in the context will reference it
address	URL	The URL to connect to in order to invoke the service
serviceClass	Class	The fully-qualified name of the interface that the bean should implement (typically, same as the service interface used on the server side)
serviceName	QName	The name of the service to invoke, if this address/WSDL hosts several. It maps to the wsdl:service@name. In the format of "ns:SERVICE_NAME" where ns is a namespace prefix valid at this scope.
endpointName	QName	The name of the endpoint to invoke, if this address/WSDL hosts several. It maps to the wsdl:port@name. In the format of "ns:ENDPOINT_NAME" where ns is a namespace prefix valid at this scope.
bindingId	URI	The URI, or ID, of the message binding for the endpoint to use. For SOAP the binding URI(ID) is specified by the JAX-WS specification. For other message bindings the URI is the namespace of the WSDL extensions used to specify the binding.
bus	Bean Reference	The bus name that will be used in the jaxws endpoint (defaults to `cxf`).
username	String	
password	String	
wsdlLocation	URL	A URL to connect to in order to retrieve the WSDL for the service. This is not required.
createdFromAPI	boolean	This indicates that the client bean was already created using jaxws API's thus at runtime when parsing the bean spring can use these values rather than the default ones. It's important that when this is true, the "name" of the bean is set to the port name of the endpoint being created in the form "{http://service.target.namespace} PortName".

It also supports many child elements:

Name	Description
jaxws:inInterceptors	The incoming interceptors for this endpoint. A list of `<bean>` or `<ref>` elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:inFaultInterceptors	The incoming fault interceptors for this endpoint. A list of `<bean>` or `<ref>` elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor

jaxws:outInterceptors	The outgoing interceptors for this endpoint. A list of <bean> or <ref> elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:outFaultInterceptors	The outgoing fault interceptors for this endpoint. A list of <bean> or <ref> elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
jaxws:features	The features that hold the interceptors for this endpoint. A list of <bean> or <ref> elements
jaxws:handlers	The JAX-WS handlers for this endpoint. A list of <bean> or <ref> elements. Each should implement javax.xml.ws.handler.Handler or javax.xml.ws.handler.soap.SOAPHandler. These are more portable than CXF interceptors, but may cause the full message to be loaded in as a DOM (slower for large messages).
jaxws:properties	A properties map which should be supplied to the JAX-WS endpoint. See below.
jaxws:dataBinding	Which DataBinding to use in the endpoint. This can be supplied using the Spring <bean class="MyDataBinding"/> syntax.
jaxws:binding	You can specify the BindingFactory for this endpoint to use. This can be supplied using the Spring <bean class="MyBindingFactory"/> syntax.
jaxws:conduitSelector	

Here is a more advanced example which shows how to provide interceptors, JAX-WS handlers, and properties:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       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.xsd
          http://cxf.apache.org/jaxws
          http://cxf.apache.org/schemas/jaxws.xsd">

  <!-- Interceptors extend e.g.
       org.apache.cxf.phase.AbstractPhaseInterceptor -->
  <bean id="anotherInterceptor" class="..." />

  <!-- Handlers implement e.g. javax.xml.ws.handler.soap.SOAPHandler -->
  <bean id="jaxwsHandler" class="..." />

  <!-- The SOAP client bean -->
  <jaxws:client id="helloClient"
                serviceClass="demo.spring.HelloWorld"
                address="http://localhost:9002/HelloWorld">
    <jaxws:inInterceptors>
      <bean class="org.apache.cxf.interceptor.LoggingInInterceptor"/>
      <ref bean="anotherInterceptor"/>
    </jaxws:inInterceptor>
    <jaxws:handlers>
      <ref bean="jaxwsHandler" />
    </jaxws:handlers>
    <jaxws:properties>
      <entry key="mtom-enabled" value="true"/>
    </jaxws:properties>
  </jaxws:client>
</beans>
```

Configuring a Spring Client (Option 2)

Building a Client using this configuration is only applicable for those wishing to inject a Client into their Spring ApplicationContext.

This approach requires more explicit Spring bean configuration than the previous option, and may require more configuration data depending on which features are used. To configure a client this way, you'll need to declare a proxy factory bean and also a client bean which is created by that proxy factory. Here is an example:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://cxf.apache.org/jaxws
      http://cxf.apache.org/schemas/jaxws.xsd">

  <bean id="proxyFactory"
    class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">
    <property name="serviceClass" value="demo.spring.HelloWorld"/>
    <property name="address" value="http://localhost:9002/HelloWorld"/>
  </bean>

  <bean id="client" class="demo.spring.HelloWorld"
    factory-bean="proxyFactory" factory-method="create"/>

</beans>
```

The JaxWsProxyFactoryBean in this case takes two properties. The service class, which is the interface of the Client proxy you wish to create. The address is the address of the service you wish to call.

The second bean definition is for the client. In this case it implements the HelloWorld interface and is created by the proxyFactory <bean> by calling the create() method. You can then reference this "client" bean and inject it anywhere into your application. Here is an example of a very simple Java class which accesses the client bean:

```
include org.springframework.context.support.ClassPathXmlApplicationContext;

public final class HelloWorldClient {

     private HelloWorldClient() { }

     public static void main(String args[]) throws Exception {
         ClassPathXmlApplicationContext context
            = new ClassPathXmlApplicationContext(
               new String[]{"my/path/to/client-beans.xml"});

         HelloWorld client = (HelloWorld)context.getBean("client");

         String response = client.sayHi("Dan");
         System.out.println("Response: " + response);
         System.exit(0);
     }
}
```

The JaxWsProxyFactoryBean supports many other properties:

Name	Description

clientFactoryBean	The ClientFactoryBean used in construction of this proxy.
password	The password which the transport should use.
username	The username which the transport should use.
wsdlURL	The wsdl URL the client should use to configure itself.
wsdlLocation	Appears to be the same as wsdlURL?
serviceName	The name of the service to invoke, if this address/WSDL hosts several. It maps to the wsdl:service@name. In the format of "ns:SERVICE_NAME" where ns is a namespace prefix valid at this scope.
endpointName	The name of the endpoint to invoke, if this address/WSDL hosts several. It maps to the wsdl:port@name. In the format of "ns:ENDPOINT_NAME" where ns is a namespace prefix valid at this scope.
inInterceptors	The incoming interceptors for this endpoint. A list of `<bean>` or `<ref>` elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
inFaultInterceptors	The incoming fault interceptors for this endpoint. A list of `<bean>` or `<ref>` elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
outInterceptors	The outgoing interceptors for this endpoint. A list of `<bean>` or `<ref>` elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
outFaultInterceptors	The outgoing fault interceptors for this endpoint. A list of `<bean>` or `<ref>` elements. Each should implement org.apache.cxf.interceptor.Interceptor or org.apache.cxf.phase.PhaseInterceptor
features	The features that hold the interceptors for this endpoint. A list of `<bean>` or `<ref>` elements
handlers	A list of `<bean>` or `<ref>` elements pointing to JAX-WS handler classes to be used for this client. Each should implement javax.xml.ws.handler.Handler or javax.xml.ws.handler.soap.SOAPHandler. These are more portable than CXF interceptors, but may cause the full message to be loaded in as a DOM (slower for large messages).
bindingConfig	
bindingId	The URI, or ID, of the message binding for the endpoint to use. For SOAP the binding URI(ID) is specified by the JAX-WS specification. For other message bindings the URI is the namespace of the WSDL extensions used to specify the binding.
bus	A reference to a CXF bus bean. Must be provided if, for example, handlers are used. May require additional Spring context imports (e.g. to bring in the default CXF bus bean).
conduitSelector	
dataBinding	Which DataBinding to use in the endpoint. This can be supplied using the Spring <bean class="MyDataBinding"/> syntax.
properties	A properties map which should be supplied to the JAX-WS endpoint.
serviceFactory	

Using some of the properties will require additional configuration in the Spring context. For instance, using JAX-WS handlers requires that you explicitly import several CXF Spring configurations, and assign the "bus" property of the JaxWsProxyFactory bean like this:

```
<import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-http.xml" />

<bean id="clientFactory" class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">
    <property name="serviceClass" value="demo.spring.HelloWorld"/>
    <property name="address" value="http://localhost:9002/HelloWorld"/>
    <property name="bus" ref="cxf" />
</bean>
```

Configuring an Endpoint/Client Proxy Using CXF APIs

JAX-WS endpoints and client proxies are implemented on top of CXF's frontend-neutral endpoint API. You can therefore use CXF APIs to enhance the functionality of a JAX-WS endpoint or client proxy, for example by adding interceptors.

To cast a client proxy to a CXF client:

```
GreeterService gs = new GreeterService();
Greeter greeter = gs.getGreeterPort();

org.apache.cxf.endpoint.Client client =
org.apache.cxf.frontend.ClientProxy.getClient(greeter);
org.apache.cxf.endpoint.Endpoint cxfEndpoint = client.getEndpoint();
cxfEndpoint.getOutInterceptors().add(...);
```

To cast a JAX-WS endpoint to a CXF server:

```
javax.xml.ws.Endpoint jaxwsEndpoint = javax.xml.ws.Endpoint.publish(
   "http://localhost:9020/SoapContext/GreeterPort", new GreeterImpl());

org.apache.cxf.jaxws.EndpointImpl jaxwsEndpointImpl =
   (org.apache.cxf.jaxws.EndpointImpl)jaxwsEndpoint;
jaxwsEndpointImpl.getOutInterceptors().add(...);
```

Configure the JAXWS Server/Client Using Spring

CXF provides <jaxws:server>, <jaxws:client> to configure the server/client side endpoint. Here are some examples:

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xmlns:soap="http://cxf.apache.org/bindings/soap"
      xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
         http://cxf.apache.org/bindings/soap
         http://cxf.apache.org/schemas/configuration/soap.xsd
         http://cxf.apache.org/jaxws
         http://cxf.apache.org/schemas/jaxws.xsd">
  <jaxws:server id="inlineImplementor"
address="http://localhost:8080/simpleWithAddress">
    <jaxws:serviceBean>
      <bean class="org.apache.hello_soap_http.GreeterImpl"/>
    </jaxws:serviceBean>
  </jaxws:server>

  <jaxws:server id="bookServer" serviceClass=
      "org.apache.cxf.mytype.AnonymousComplexTypeImpl"
    address="http://localhost:8080/act"
    bus="cxf">
    <jaxws:invoker>
      <bean class="org.apache.cxf.service.invoker.BeanInvoker">
        <constructor-arg>
          <bean class="org.apache.cxf.mytype.AnonymousComplexTypeImpl"/>
        </constructor-arg>
      </bean>
    </jaxws:invoker>
    <jaxws:dataBinding>
      <bean class="org.apache.cxf.jaxb.JAXBDataBinding">
        <property name="namespaceMap">
          <map>
             <entry>
               <key>
                 <value>http://cxf.apache.org/anon_complex_type/</value>
               </key>
               <value>BeepBeep</value>
             </entry>
          </map>
        </property>
      </bean>
    </jaxws:dataBinding>
  </jaxws:server>

  <jaxws:client id="bookClient"
    serviceClass="org.apache.cxf.mytype.AnonymousComplexType"
    address="http://localhost:8080/act"/>

</beans>
```

Since JAX-WS frontend server and client spring configuration parser are inherited from the simple frontend, please see Simple Frontend Configuration for the attribute and element definitions.

Configure the JAXWS Server Using SpringBoot

Here is an example:

```
import org.apache.cxf.Bus;
import org.apache.cxf.jaxws.EndpointImpl;
import org.apache.cxf.transport.servlet.CXFServlet;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.embedded.EmbeddedServletContainerFactory;
import org.springframework.boot.context.embedded.ServletRegistrationBean;
import org.springframework.boot.context.web.SpringBootServletInitializer;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.ImportResource;

@Configuration
@EnableAutoConfiguration
@ImportResource({ "classpath:META-INF/cxf/cxf.xml" })
public class Application extends SpringBootServletInitializer {

    @Autowired
    private ApplicationContext applicationContext;

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    // Replaces the need for web.xml
    @Bean
    public ServletRegistrationBean servletRegistrationBean(ApplicationContext context)
{
        return new ServletRegistrationBean(new CXFServlet(), "/api/*");
    }

    // Replaces cxf-servlet.xml
    @Bean
    // <jaxws:endpoint id="helloWorld"
implementor="demo.spring.service.HelloWorldImpl" address="/HelloWorld"/>
    public EndpointImpl helloService() {
        Bus bus = (Bus) applicationContext.getBean(Bus.DEFAULT_BUS_ID);
        Object implementor = new HelloWorldImpl();
        EndpointImpl endpoint = new EndpointImpl(bus, implementor);
        endpoint.publish("/hello");
        return endpoint;
    }


    // Used when deploying to a standalone servlet container, i.e. tomcat
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application)
{
        return application.sources(Application.class);
    }

}
```