# Using Celtix Configuration

## Table of Contents



## Overview

This document describes how to write and use a Celtix configuration file.

A Celtix configuration file is actually a Spring Framework XMLBeanFactory configuration file (see `http://www.springframework.org/docs/reference/beans.html#beans-factory` and `http://www.springframework.org/`). However, other than learning how to write the configuration file, you do not need to know anything about the Spring Framework or its APIs.

Illustration 1 shows the hierarchical organization of configurable components within Celtix. The shaded components may be configured through use of a Celtix configuration file. Note that within the client application, configuration settings may be applied at the level of the service (that is, to all invocations that use a specific proxy instance), port, or transport. Within the server application, configuration settings may be applied at the level of the endpoint (that is, to all invocations against a specific service) or transport.

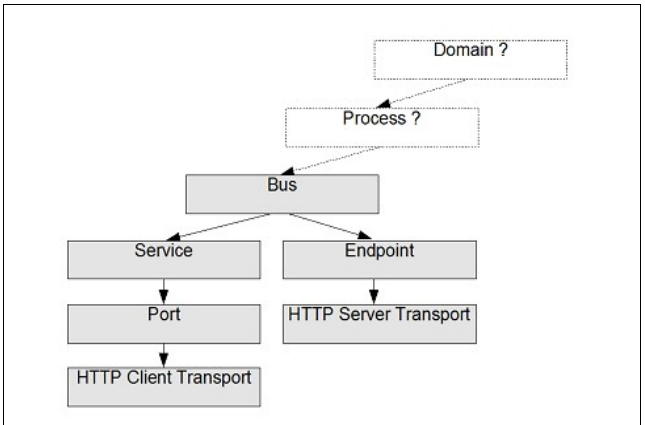


*Illustration 1: Configuration Hierarchy*

In a later section of this document, you will see how an understanding of this hierarchy is used to identify the component to which a collection of configuration settings should be applied.

Celtix also has some fully implemented services, for example, the routing service, that require configuration. These services are also configured using the approach described in this document. In a configuration file used by a Celtix service, the identity of the configurable component is not derived from the configuration hierarchy illustrated in Illustration 1.

This document is based on the Celtix v1.0 General Availability release (April 2006), so there may be revisions in interim builds and subsequent releases. It is essential that you refer to the metadata XML and schema files that are packaged with your installation, see the following section The Celtix Metadata XML and Schema Configuration Files, when writing or editing Celtix configuration files.

Refer to the Celtix configuration Wiki page for the most current description of this functionality. This page is located at: (`https://wiki.objectweb.org/celtix/Wiki.jsp?page=ConfigurationDocumentation`).

## Structure of a Celtix Configuration File

A Celtix configuration file has the syntax shown in the following fragment:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM
      "http://celtix.objectweb.org/configuration/spring/celtix-spring-beans.dtd">

<beans xmlns:...>

  <bean id="..." abstract="true">
      <property name="...">
          <value>
              ...
          </value>
      </property>
  </bean>

  <bean id="..." class="..." parent="...">
      <property name="...">
          <value>
              ...
          </value>
      </property>
  </bean>

  <bean id="..." class="...">
      <property name="...">
          <value>
              ...
          </value>
      </property>
  </bean>

</beans>
```

With the exception of the URL listed in the `DOCTYPE` declaration, this syntax derives directly from the Spring Framework XMLBeanFactory configuration file. Starting from this basic framework, your task is to fill in the entries indicated by the ellipsis (...).

In the opening `<beans>` element, you will include namespace declarations for the schema files that define acceptable configuration entries. The file may then include one, or more, `<bean>` elements, each of which corresponds to a configurable component within Celtix.

The `id` attribute represents the component to which the configuration will be applied, for example, a Bus, a service, a port, a transport, or a Celtix service. The value you supply for the `id` attribute generally includes a Bus identifier and the name of the service, or the service and port, to which the configuration applies, that is, the value of the `id` attribute is derived from the configuration hierarchy. However, the `id` value is nothing more than a unique string, so there are exceptions to this rule, such as configuration entries for the Celtix routing service, which will be discussed in a later section of this document. The `class` attribute provides the class name of a bean within the Spring Framework infrastructure that is responsible for managing the configuration entries.

Note that there are three ways of specifying a `<bean>` element. The first approach is used to define an abstract configuration, that is, a configuration that will be reused in another `<bean>` element. Since this declaration only describes configuration entries that will be reused in another declaration, there is no corresponding `class` entry. The second approach is used to define a `<bean>` that reuses the configuration entries defined in an abstract `<bean>` element. The value of the parent attribute is the `id` of the abstract `<bean>`. And the third approach is used to completely define a `<bean>`.

Finally, the `property` element corresponds to a configurable variable, identified by a `name` attribute, and the content within the `<value>` element is the value for that configuration entry.

Filling in values for the `id` attribute and the `class` attribute is fairly straight-forward. The information needed to create an `id` that indicates the configurable component can be derived from the WSDL file for the service (although the `id` can be any unique string), whereas the information needed to create the class name is derived from the namespace URI of the metadata XML file that defines acceptable configuration entries for the component being configured.

To complete the information in the `<propertt>` element and the `<value>` elements you use the types that are defined in multiple schema and metadata XML files, which are described in the following section.

# The Celtix Metadata XML and Schema Configuration Files

For each configurable component, Celtix provides an XML file that contains the metadata needed to identify and set a configuration variable, and a schema file that defines types typically used in configuration entries. In both the Celtix binary and source distributions, copies of these files exist within the `celtix.jar` file located in the directory `CELTIX_HOME/lib` and in the `CELTIX_HOME/resources` directory.

When you are actually writing a Celtix configuration file you must refer to the copies of these files that ship with your actual product to be certain that you have correctly coded each element. The best place to start is with an existing configuration file, which you can modify to suit the requirements of your application. If you need to define another configuration entry, you will first use the metadata XML file to determine the name and content of the `<property>` element and then use a schema file to determine the content of the `<value>` element.

## *The Celtix XML Metadata Files*

The content of each of these XML metadata files adheres to the structure defined in the schema file `resources/schemas/configuration/metadata.xsd`. Each `<configItem>` element represents a configurable variable for the component.

The file `resources/config-metadata/bus-config.xml` defines the metadata for configuring the Bus component. This file describes three configurable variables: `bindingFactories`, `transportFactories`, and `resourceResolvers`. Note that the namespace assigned to this file's content is `http://celtix.objectweb.org/bus/bus-config`. When writing a configuration file that includes Bus related configuration entries, the value of the `class` attribute in the `<bean>` element is derived from this namespace declaration: `org.objectweb.celtix.bus.bus_config.spring.BusConfigBean`.

The file **resources/config-metadata/endpoint-config.xml** defines the metadata for configuring the endpoint (server) component. This file describes four configurable variables: **handlerChain**, **systemHandlerChain**, **serverContextInspectors**, and **enableSchemaValidation**. Note that the namespace assigned to this file's content is **http://celtix.objectweb.org/bus/jaxws/endpoint-config**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.jaxws.endpoint_config.spring.EndpointConfigBean**.

The file **resources/config-metadata/http-client-config.xml** defines the metadata for configuring the service (client) HTTP transport. This file describes four configurable variables: **httpClient**, **authorization**, **proxyAuthorization**, and **ssl**. Note that the namespace assigned to this file's content is **http://celtix.objectweb.org/bus/transports/http/http-client-config**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.transports.http.http_client_config.spring.HttpClientConfigBean**.

The file **resources/config-metadata/http-listener-config.xml** defines the metadata for configuring an HTTP transport listener. This file describes two configurable variables: **httpListener** and **ssl**. Note that the namespace assigned to this file's content is **http://celtix.objectweb.org/bus/transports/http/http-listener-config**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.transports.http.http_server_config.spring.HttpListenerConfigBean**.

The file **resources/config-metadata/http-server-config.xml** defines the metadata for configuring the endpoint (server) HTTP transport. This file describes three configurable variables: **httpServer**, **authorization**, and **ssl**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.transports.http.http_server_config.spring.HttpServerConfigBean**.

The file **resources/config-metadata/instrumentation-config.xml** defines the metadata for configuring instrumentation. This file describes two configurable variables: **InstrumentationControl** and **MBServer**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.instrumentation.spring.InstrumentationConfigBean**.

The file **resources/config-metadata/jms-client-config.xml** defines the metadata for configuring the service (client) JMS transport. This file describes two configurable variables: **jmsClient** and **jmsAddress**. Note that the namespace assigned to this file's content is **http://celtix.objectweb.org/bus/transports/jms/jms-client-config**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.transports.jms.jms_client_config.spring.JmsClientConfigBean**.

The file **resources/config-metadata/jms-server-config.xml** defines the metadata for configuring the endpoint (server) JMS transport. This file describes three configurable variables: **jmsServer** and **jmsAddress**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.transports.jms.jms_server_config.spring.JmsServerConfigBean**.

The file **resources/config-metadata/port-config.xml** defines the metadata for configuring the endpoint's port component. This file describes five configurable variables: **address**, **bindingId**, **transportId**, **handlerChain**, **systemHandlerChain**, and **enableSchemaValidataion**. Note that the namespace assigned to this file's content is **http://celtix.objectweb.org/bus/jaxws/port-config**. When writing a configuration file that includes endpoint related configuration entries, the value of the **class** attribute in the **<bean>** element is: **org.objectweb.celtix.bus.jaxws.port_config.spring.PortConfigBean**.

The file `resources/config-metadata/rm-config.xml` defines the metadata for configuring WS-Reliable Messaging (WS-RM). This file describes three configurable variables: `rmAssertion`, `sourcePolicies`, and `destinationPolicies`. Note that the namespace assigned to this file's content is: `http://celtix.objectweb.org/bus/ws/rm/rm-config`. When writing a configuration file that includes WS-RM related configuration entries, the value of the class attribute in the `<bean>` element is: `org.objectweb.celtix.bus.ws.rm.rm_config.spring.RmConfigBean`.

The file `resources/config-metadata/router-config.xml` defines the metadata for configuring the Celtix routing service. This file describes one configurable variable: `routesWSDL`. Note that the namespace assigned to this file's content is: `http://celtix.objectweb.org/routing/configuration`. When writing a configuration file that includes routing configuration entries, the value of the class attribute in the `<bean>` element is: `org.objectweb.celtix.routing.configuration.spring.ConfigurationBean`.

The file `resources/config-metadata/service-config.xml` is a placeholder for future service level configuration settings. It currently has no content.

## The Celtix Schema Files

Your jumping off point for specifying a configuration entry is the metadata XML file where you determine what configuration variables you can include and identify which schema files contain the corresponding type information. Also, use the namespace declaration in the metadata XML file to determine the name of the Spring configuration bean corresponding to the component you want to configure. Then you turn to the schema file to discover the syntax and type of data you need to add to the configuration file. In some cases, you will need to refer to more than one schema file to complete your configuration entry. In this case, the metadata XML file will include namespace declarations for all of the associated schema files. The schema files are located in both the `resources/schemas/configuration` and `resources/schemas/wsdl` subdirectories.

A simple example will make this clearer.

## Configuring the Routing Service

Review the content of the metadata XML file `router-config.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<cm:config
    xmlns:cm="http://celtix.objectweb.org/configuration/metadata"
    xmlns:router-conf="http://celtix.objectweb.org/routing/configuration"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    namespace="http://celtix.objectweb.org/routing/configuration">

    <cm:configImport
        namespace="http://celtix.objectweb.org/routing/configuration"
        location="schemas/wsdl/routing.xsd"/>

    <cm:configItem>
        <cm:name>routesWSDL</cm:name>
        <cm:type>router-conf:urlListPolicy</cm:type>
        <cm:description>
            List of wsdl urls used by router
        </cm:description>
        <cm:lifecyclePolicy>bus</cm:lifecyclePolicy>
    </cm:configItem>

</cm:config>
```

You can extract three pieces of information from this file (highlighted in blue).

1. The namespace and location of the schema file(s) that define the types used to specify the configurable variables.
2. The namespace assigned to the metadata XML file, which leads to the class name of the Spring configuration bean.
3. The name and type of the configurable variable.

Using this information, you can begin writing the configuration file with the following content.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM "http://celtix.objectweb.org/configuration/spring/celtix-
spring-beans.dtd">

<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types"
    xmlns:router-conf="http://celtix.objectweb.org/routing/configuration">

  <bean id="..."
     class="org.objectweb.celtix.routing.configuration.spring.ConfigurationBean">

      <property name="routesWSDL">
          <value>
              ...
          </value>
      </property>
  </bean>

</beans>
```

The value you assign to the `<bean>` element's id attribute needs to be a unique string. Since this configuration file will be used by the Celtix routing service, and not specifically by one of the components identified in Illustration 1, you do not derive the id from the Bus identifier. You can simply specify a string value.

To complete the `<value>` element, you must refer to the associated schema file. The schema file `resources/schemas/wsdl/routing.xsd` is defined within the `http://celtix.objectweb.org/routing/configuration` namespace, and this is the file you use to define the content of the `<value>` element.

This schema file contains multiple type definitions. Most of these types are used in defining a `<route>` specification within a WSDL file, and do not relate to configuring the routing service. The types `wsdlUrl` and `urlListPolicy` (at the end of the file) are relevant to configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   ...
   xmlns:r="http://celtix.objectweb.org/routing/configuration"
   targetNamespace="http://celtix.objectweb.org/routing/configuration">

    ...

    <xs:element name="wsdlUrl" type="r:urlListPolicy"/>

    <xs:complexType name="urlListPolicy">
        <xs:sequence>
            <xs:element name="url" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

</schema>
```

In the metadata XML file `router-config.xml`, the type for the `routesWSDL` configurable variable is given as `urlListPolicy`, which is a sequence of string entries, where each entry is the path to a WSDL file that includes a `<route>` definition. In the schema file `routing.xsd` you can see that the element `<wsdlUrl>` wraps the `urlListPolicy` type. So to complete the configuration entry, you nest a `<wsdlUrl>` element, which in turn contains one or more `<url>` elements, under the `<value>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM "http://celtix.objectweb.org/configuration/spring/celtix-
spring-beans.dtd">

<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types"
    xmlns:router-conf="http://celtix.objectweb.org/routing/configuration">

  <bean id="..."
     class="org.objectweb.celtix.routing.configuration.spring.Configurationbean">

      <property name="routesWSDL">
          <value>
              <router-conf:wsdlUrl>
                  <router-conf:url>/wsdl/router.wsdl</router-conf:url>
              </router-conf:wsdlUrl>
          </value>
      </property>
  </bean>

</beans>
```

Note how the configuration entry uses the elements `<wsdlUrl>` and `<url>` and not the type `urlListPolicy`. A more involved example is presented in the section Writing a Celtix Configuration File.

# Writing a Celtix Configuration File

Review the structure of a Celtix configuration file (see page 2). You will write a separate configuration file for each process that you want to configure. You will then define a `<bean>` element for each component that requires configuration, which means that you must provide a value for the `id` and `name` attributes in the `<bean>` element, and for the `name` attribute in one, or more, `<property>` elements. Within the `<property>` element, you specify a value for this configurable entry. Entering the value is the most difficult part of this process as you must use the information in the schema and XML metadata files as guides to the proper syntax.

## *The Namespace Declarations*

Within the opening `<beans>` element you should include namespace declarations corresponding to the schema files `org/objectweb/celtix/configuration/config-metadata/types.xsd` and `org/objectweb/celtix/configuration/config-metadata/metadata.xsd`. These declarations are then available to all `<bean>` elements within the configuration file. The beginning of each Celtix configuration file is shown in the following fragment.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM
      "http://celtix.objectweb.org/configuration/spring/celtix-spring-beans.dtd">

<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types"
    xmlns:jaxws-types="http://celtix.objectweb.org/bus/jaxws/configuration/types">
```

## The class Attribute

In the section The Celtix XML Metadata Files, you saw how the namespace used in the metadata XML file is mapped to the name of a Java bean class within the Spring Framework. This class name is then used as the value of the `class` attribute within a `<bean>` element. The information in the metadata XML and Celtix configuration files is used by the Celtix runtime to instantiate and initialize a bean instance that will manage a component's configuration. The following table summarizes the class attribute values that correspond to each configurable component.

| Component | Bean Class Name |
|---|---|
| Bus | `org.objectweb.celtix.bus.bus_config.spring.BusConfigBean` |
| Endpoint | `org.objectweb.celtix.bus.jaxws.endpoint_config.spring.EndpointConfigBean` |
| Port | `org.objectweb.celtix.bus.jaxws.port_config.spring.PortConfigBean` |
| HTTP Client Transport | `org.objectweb.celtix.bus.transports.http.http_client_config.spring.HttpClientConfigBean` |
| HTTP Server Transport | `org.objectweb.celtix.bus.transports.http.http_server_config.spring.HttpServerConfigBean` |
| HTTP Listener | `org.objectweb.celtix.bus.transports.http.http_listener_config.spring.HttpListenerConfigBean` |
| JMS Client Transport | `org.objectweb.celtix.bus.transports.jms.jms_client_config.spring.JmsServerConfigBean` |
| JMS Server Transport | `org.objectweb.celtix.bus.transports.jms.jms_server_config.spring.JmsServerConfigBean` |
| Instrumentation | `org.objectweb.celtix.bus.instrumentation.spring.InstrumentationConfigBean.` |
| Reliable Messaging | `org.objectweb.celtix.bus.ws.rm.rm_config.spring.RmConfigBean` |
| Routing Service | `org.objectweb.celtix.routing.configuration.spring.ConfigurationBean` |

## The id Attribute

While the value of the `id` attribute can indicate what component the configuration applies to, it is not a simple one-to-one relationship as is the `class` attribute. This is because the id is generally derived from information in the WSDL file that describes the service. This can be illustrated using the simple hello world WSDL file that follows. The entries you need to specify the `id` are indicated in bold face font: the targetNamespace, the service name, and the port name. You also need the name assigned to the Celtix Bus. By default, in a simple application where the Bus instance is created transparently by the Celtix runtime, the Bus name is `celtix`.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="HelloWorld"
    targetNamespace="http://objectweb.org/hello_world"
    ...>
    <wsdl:types>
        <schema targetNamespace=
```

```
                "http://objectweb.org/hello_world_soap_http/types"
                ...>
            ...
        </schema>
    </wsdl:types>

    <wsdl:message name=...>
        ...
    </wsdl:message>
      .
      .
      .
    <wsdl:message name=...>
        ...
    </wsdl:message>

    <wsdl:portType name=...>
        <wsdl:operation name=...>
            <wsdl:input message=... name=.../>
            <wsdl:output message=... name=.../>
        </wsdl:operation>
          .
          .
          .
        <wsdl:operation name=...>
            <wsdl:input message=... name=.../>
            <wsdl:output message=... name=.../>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name=... type=...>
        <soap:binding style="document"
                      transport="http://schemas.xmlsoap.org/soap/http"/>

        <wsdl:operation name=...>
            <soap:operation soapAction="" style="document"/>
            <wsdl:input name=...>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name=...>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
          .
          .
          .
        <wsdl:operation name=...>
            <soap:operation soapAction="" style="document"/>
            <wsdl:input name=...>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name=...>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="SOAPService">
        <wsdl:port binding=... name="SoapPort">
            <soap:address location="http://localhost:9000/SoapContext/SoapPort"/>
```

```
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

The following table lists which pieces of information are needed to create the `id` for each configurable component and shows the resulting value specific to this WSDL file.

| *Component* | *Instance Identifier* | *id Value* |
|---|---|---|
| Bus | Bus name | `celtix` |
| Endpoint | Bus name and QName of service | `celtix.{http://objectweb.org/hello_world}SOAPService` |
| Port | Bus name, QName of service, and name of port | `celtix.{http://objectweb.org/hello_world}SOAPService/SoapPort` |
| HTTP Client Transport | Bus name, QName of service, name of port, and string constant | `celtix.{http://objectweb.org/hello_world}SOAPService/SoapPort.http-client` |
| HTTP Server Transport | Bus name, QName of service, name of port, and string constant | `celtix.{http://objectweb.org/hello_world}SOAPService/SoapPort.http-server` |
| HTTP Listener | Bus name, string constant, and port number (where port number is the TCP/IP port number set in the server mainline code). | `celtix.http-listener.port_number` |

## The property Element

Perhaps the best way to learn how to complete the rest of the Celtix configuration file is to look as some examples.

## Setting the URL through configuration

The most straight-forward example is the situation in which you want configure the client application so that the URL used to invoke on the endpoint is defined in the configuration file rather than through the WSDL file's content. For the simple hello world example described above, the following fragment shows the corresponding `<property>` element, which corresponds to the configuration entry that will set the URL.

```
<bean id="celtix.{http://objectweb.org/hello_world}SOAPService.SoapPort"
      class="org.objectweb.celtix.bus.jaxws.port_config.spring.PortConfigBean">

      <property name="address">
          <value>
              <ct:stringValue>
                http://localhost:9002/SoapContext/SoapPort
              </ct:stringValue>
          </value>
      </property>

</bean>
```

Since you are configuring an HTTP port, the appropriate XML metadata file to use as a guide is `port-config.xml`, and the desired `configItem` element is `address`. Assign the value of the configItem's `name` attribute to the `<property>` element's `name` attribute. Within the `<value>` element, you will enter a string that is the desired URL. Notice that the element used to delimit the URL correspond to an element type defined in the schema file `std-types.xsd`. The configuration scheme also supports a shorthand notation that eliminates the `<stringValue>` tags.

```
<value>http://localhost:9002/SoapContext/SoapPort</value>
```

## Specifying a Handler Through Configuration

A more complex example is contained in the server configuration file, `celtix-server.xml`, which is included in the handlers product demo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM
       "http://celtix.objectweb.org/configuration/spring/celtix-spring-beans.dtd">

<beans
   xmlns:ct="http://celtix.objectweb.org/configuration/types"
   xmlns:jaxws-types="http://celtix.objectweb.org/bus/jaxws/configuration/types">

  <bean id="celtix.{http://www.objectweb.org/handlers}AddNumbersService"
        class=
      "org.objectweb.celtix.bus.jaxws.endpoint_config.spring.EndpointConfigBean">

      <property name="handlerChain">
          <value>
              <jaxws-types:handler-chain>
                 <jaxws-types:handler>
                      <jaxws-types:handler-name>
                          File Logging Handler
                      </jaxws-types:handler-name>
                      <jaxws-types:handler-class>
                          demo.handlers.common.FileLoggingHandler
                      </jaxws-types:handler-class>
                 </jaxws-types:handler>
              </jaxws-types:handler-chain>
          </value>
```

```
    </property>

  </bean>
</beans>
```

Since you are configuring a service endpoint, the appropriate XML metadata file to use as a guide is `endpoint-config.xml`, and the desired `<configItem>` element is handlerChain. Assign the value of the configItem's `name` attribute to the `<property>` element's `name` attribute. Within the `<value>` element, you will make entries that identify the class within your application that implements the handler you want to deploy.

Deciding what to place within the `<value>` element is a little more complex than in the previous example. Look at the metadata XML file and note that the `handlerChain` entry corresponds to the `handlerChainType` defined in the schema file `jaxws-types.xsd`. However, since the `handlerChainType` is a complex type, and the configuration file must include an element type, you identify the `<handler-chain>` element type as a suitable replacement. Then you look up the composition of the `handlerChainType` and determine that it includes a optional handler-name, of type string, and a sequence of zero or more `handlerType` instances.

```
    <xs:complexType name="handlerChainType">
        <xs:sequence>
            <xs:element name="handler-chain-name" type="xs:string"
                        minOccurs="0"/>
            <xs:element name="handler" type="tns:handlerType"
                        minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
```

And the `handlerType` is also a sequence in which only the second element is required.

```
    <xs:complexType name="handlerType">
        <xs:sequence>
            <xs:element name="handler-name" type="xs:string"
                        minOccurs="0"/>
            <xs:element name="handler-class" type="xs:string"/>
            <xs:element name="init-param" type="tns:handlerInitParamType"
                        nillable="false" minOccurs="0"
                        maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
```

In the configuration file, use the handler-name and handler-class types to specify the handler class.

```
      <value>
          <jaxws-types:handler-chain>
             <jaxws-types:handler>
                  <jaxws-types:handler-name>
                      File Logging Handler
                  </jaxws-types:handler-name>
                  <jaxws-types:handler-class>
                      demo.handlers.common.FileLoggingHandler
                  </jaxws-types:handler-class>
             </jaxws-types:handler>
          </jaxws-types:handler-chain>
      </value>
```

What if your handler class requires initialization parameters. The `handlerType` allows for this through its `<init-param>` element, which is an instance of `handlerInitParamType`. The `handlerInitParamType` is a sequence of name value pairs that correspond the handler's initialization values. Since the `handlerInitParamType` is an unbounded sequence, your configuration entry may have as many `<init-param>` elements as required. The following fragment illustrates how the `<init-param>` elements would be included in the configuration file.

```
    <property name="handlerChain">
```

```
        <value>
            <jaxws-types:handler-chain>
               <jaxws-types:handler>
                    <jaxws-types:handler-name>...</jaxws-types:handler-name>
                    <jaxws-types:handler-class>...</jaxws-types:handler-class>

                    <jaxws-types:init-param>
                        <jaxws-types:param-name>
                            arg1
                        </jaxws-types:param-name>
                        <jaxws-types:param-value>
                            value1
                        </jaxws-types:param-value>
                    </jaxws-types:init-param>

                    <jaxws-types:init-param>
                        <jaxws-types:param-name>
                            arg2
                        </jaxws-types:param-name>
                        <jaxws-types:param-value>
                            value2
                        </jaxws-types:param-value>
                    </jaxws-types:init-param>

                </jaxws-types:handler>
            </jaxws-types:handler-chain>
        </value>
    </property>
```

## Setting Authorization Information

Since you are configuring the client service, the appropriate metadata XML file to use as a guide is **http-client-config.xml**, and the desired **<configItem>** element is **authorization**. Assign the value of the configItem's **name** element to the **<property>** element's **name** attribute.

Look at the metadata XML file and note that the **authorization** entry corresponds to the **AuthorizationPolicy** type defined in the schema file **security.xsd**. However, since the **AuthorizationPolicy** type is a complex type, and the configuration file must include a element type, you identify the **authorization** element as a suitable replacement. Then you look up the composition of the **AuthorizationPolicy** type and determine that it is a sequence of elements: **UserName**, **Password**, **AuthorizationType**, and **Authorization**.

```
<xs:complexType name="AuthorizationPolicy">
    <xs:sequence>
        <xs:element name="UserName" type="xs:string" minOccurs="0"/>

        <xs:element name="Password" type="xs:string" minOccurs="0"/>

        <xs:element name="AuthorizationType" type="xs:string" minOccurs="0"/>

        <xs:element name="Authorization" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:element name="authorization" type="tns:AuthorizationPolicy"/>
```

Combining this information, leads to the following **<bean>** element. Note the use of the **sec:** namespace prefix. You must include the corresponding namespace declaration at the beginning of the configuration file.

```
<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types"
    xmlns:sec="http://celtix.objectweb.org/bus/configuration/security">

  <bean id="celtix.{http://objectweb.org/hello_world_soap_http}SOAPService.
                                                        SoapPort.http-client"
        class="org.objectweb.celtix.bus.transports.http.http_client_config.
                                                    spring.HttpClientConfigBean">

     <property name="authorization">
         <value>
             <sec:authorization>
                 <sec:UserName>User</sec:UserName>
                 <sec:Password>celtix</sec:Password>
             </sec:authorization>
         </value>
     </property>

  </bean>
</beans>
```

## Setting Transport Attributes

The XML metadata file `http-client-config.xml`, using the `<configItem>` element `httpClient`, indicates that the `HTTPClientPolicy` type can be used to set transport attributes. The `HTTPClientPolicy` type, defined in the schema file `resources/schemas/wsdl/http-conf.xsd`, is a complex type consisting of multiple attributes. This example illustrates how to use attributes to send the request to a proxy server.

```
    <xs:complexType name="HTTPClientPolicy">
        <xs:annotation>
            <xs:documentation>HTTP client configuration properties.
            Used for configuring a HTTP client port.
            </xs:documentation>
        </xs:annotation>

        <xs:complexContent>
            <xs:extension base="wsdl:tExtensibilityElement">

                <!-- Other attribute definitions -->

                <xs:attribute name="AutoRedirect" type="xs:string" use="optional"
                              default="false"/>

                <!--Proxy server attributes-->
                <xs:attribute name="ProxyServer" type="xs:string" use="optional">
                    <xs:annotation>
                        <xs:documentation>
                            Address of proxy server, if used
                            (proxy servers are a special kind of firewall)
                            proxy.mycompany.com
                        </xs:documentation>
                    </xs:annotation>
                </xs:attribute>

                <xs:attribute name="ProxyServerPort" type="xs:int"
                              use="optional">
                    <xs:annotation>
                        <xs:documentation>
                            Port number of proxy server.
```

```
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>

            <xs:attribute name="ProxyServerType"
                          type="http-conf:proxyServerType"
                          use="optional" default="HTTP">
                <xs:annotation>
                    <xs:documentation>
                        Type of number of proxy server.
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>

        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="client" type="http-conf:HTTPClientPolicy"/>
```

The `<client>` element may be used to reference the `HTTPClientPolicy` type in a Celtix configuration file. Note that the bean class used to configure transport attributes is the same as the bean class used to configure authorization. The `http-conf.xsd` schema file is described in the namespace `http://celtix.objectweb.org/transports/http/configuration`, and you must include a prefix definition for this namespace at the beginning of the Celtix configuration file.

```
<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types"
    xmlns:sec="http://celtix.objectweb.org/bus/configuration/security"
    xmlns:http-conf="http://celtix.objectweb.org/transports/http/configuration">

  <bean id="celtix.{http://objectweb.org/hello_world_soap_http}SOAPService.
                                                        SoapPort.http-client"
        class="org.objectweb.celtix.bus.transports.http.http_client_config.
                                                    spring.HttpClientConfigBean">

      <property name="httpClient">
          <value>
              <http-conf:client ProxyServer="localhost" ProxyServerPort="5049"
                                AutoRedirect="true"/>
          </value>
      </property>

  </bean>
</beans>
```

## An Example Application

The Celtix product includes several sample applications that illustrate configuration techniques. However, the best way to learn is to try it yourself, so here is a simple example. Besides giving you a chance to write a Celtix configuration file, this example will show you how to direct your application to actually use the configuration file.

## The Application Code

For this example, you will build on the hello_world product demo. Build this demo and confirm that it runs successfully using both the `ant` utility and the `java` executable.

In this demo, the server mainline code sets the URL on which the application will listen for incoming requests.

```
package demo.hw.server;
import javax.xml.ws.Endpoint;

public class Server {

    protected Server() throws Exception {
        System.out.println("Starting Server");

        Object implementor = new GreeterImpl();
        String address = "http://localhost:9000/SoapContext/SoapPort";
        Endpoint.publish(address, implementor);
    }

    public static void main(String args[]) throws Exception {
        new Server();
        System.out.println("Server ready...");

        Thread.sleep(5 * 60 * 1000);
        System.out.println("Server exiting");
        System.exit(0);
    }
}
```

The client, however, obtains the URL from the WSDL file.

```
    <wsdl:service name="SOAPService">
        <wsdl:port binding="tns:Greeter_SOAPBinding" name="SoapPort">
            <soap:address location="http://localhost:9000/SoapContext/SoapPort"/>
        </wsdl:port>
    </wsdl:service>
```

In a text editor, open the WSDL file and change the TCP/IP port (any value is acceptable, just be certain that it is an unused port number). Save the file.

```
            <soap:address location="http://localhost:9002/SoapContext/SoapPort"/>
```

Now if you try to run the client, it will be unable to contact the server and the invocation requests will fail.

## The Configuration File

Write a configuration file that sets the address property of the PortConfigBean. (NOTE: In the `id` attribute, remove the line break that follows `=`. This was added only for legibility in this document.)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM
     "http://celtix.objectweb.org/configuration/spring/celtix-spring-beans.dtd">

<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types">

  <bean id=
        "celtix.{http://objectweb.org/hello_world_soap_http}SOAPService.SoapPort"
        class="org.objectweb.celtix.bus.jaxws.port_config.spring.PortConfigBean">

      <property name="address">
          <value>
              <ct:stringValue>
                http://localhost:9000/SoapContext/SoapPort
              </ct:stringValue>
          </value>
      </property>
```

```
  </bean>

</beans>
```

Alternatively, you may code the `value` element as:

```
          <value>
                http://localhost:9000/SoapContext/SoapPort
          </value>
```

Next, add a `<bean>` element for a HttpClientConfigBean and include a `<property>` element to set authorization details and another `<property>` element to set the attributes related to the proxy server.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM
     "http://celtix.objectweb.org/configuration/spring/celtix-spring-beans.dtd">

<beans
    xmlns:ct="http://celtix.objectweb.org/configuration/types"
    xmlns:sec="http://celtix.objectweb.org/bus/configuration/security"
    xmlns:http-conf="http://celtix.objectweb.org/transports/http/configuration">

  <bean id=
        "celtix.{http://objectweb.org/hello_world_soap_http}SOAPService.SoapPort"
        class="org.objectweb.celtix.bus.jaxws.port_config.spring.PortConfigBean">

      <property name="address">
          <value>
              <ct:stringValue>
                http://localhost:9000/SoapContext/SoapPort
              </ct:stringValue>
          </value>
      </property>
  </bean>

  <bean id="celtix.{http://objectweb.orb/hello_world_soap_http}SOAPService.
                                                          SoapPort.http-client"
        class="org.objectweb.celtix.bus.transports.http.http_client_config.
                                                  spring.HttpClientConfigBean">

      <property name="authorization">
          <value>
              <sec:authorization>
                  <sec:UserName>User</sec:UserName>
                  <sec:Password>celtix</sec:Password>
              </sec:authorization>
          </value>
      </property>

      <!-- If you have a proxy server, remove comments and edit the values of
           ProxyServer and ProxyServerPort accordingly. -->
   <!--
      <property name="httpClient">
          <value>
              <http-conf:client ProxyServer="localhost" ProxyServerPort="5049"
                                AutoRedirect="true"/>
          </value>
      </property>
   -->

  </bean>
```

```
</beans>
```

The configuration file now includes two **<bean>** elements; the first represents configurable settings described in the **port-config.xml** metadata file, and the second represents configurable settings in the **http-client-config.xml** metadata file.

Save this file in text format into the directory ***<installationDirectory>*/celtix/samples/hello_world**; you may give the file any name; the next section assumes that the file is saved as **client.xml**. The WSDL file now has an incorrect URL while the configuration file and server mainline have the same URL.

## Running the Example

Build the applications by issuing the **ant build** command. Then, using either the **ant** utility or the **java** executable, start the server application as described in the README.

When running the client application, you must force it to read the configuration file. You do this by providing a **-D** command line argument to the **java** executable.

To run the client using the **java** executable, enter the command:

```
java -Djava.util.logging.config.file=%CELTIX_HOME%\etc\logging.properties
     -Dceltix.config.file=file:///%CELTIX_HOME%\samples\hello_world\client.xml
     demo.hw.client.Client .\wsdl\hello_world.wsdl
```

To run the client using the **ant** utility, you will need to edit the **build.xml** file that is in the directory ***<installationDirectory>*/celtix/samples/hello_world**. Open this file in a text editor and modify the client target.

```
    <target name="client" description="run demo client">
        <property name="param" value=""/>
        <celtixrun classname="demo.hw.client.Client"
                   jvmarg1="-Dceltix.config.file=file:///${basedir}/client.xml"
                   param1="${basedir}/wsdl/hello_world.wsdl"
                   param2="${op}" param3="${param}"/>
    </target>
```