Using Celtix Configuration

Table of Contents

Overview

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

A Celtix configuration file is 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 in Celtix. The shaded components can be configured using a Celtix configuration file. In a client application, configuration settings may be applied at the level of a service (to all invocations using a specific proxy instance), a port, or a transport. In a server application, configuration settings may be applied at the level of an endpoint (to all invocations against a specific service) or transport.

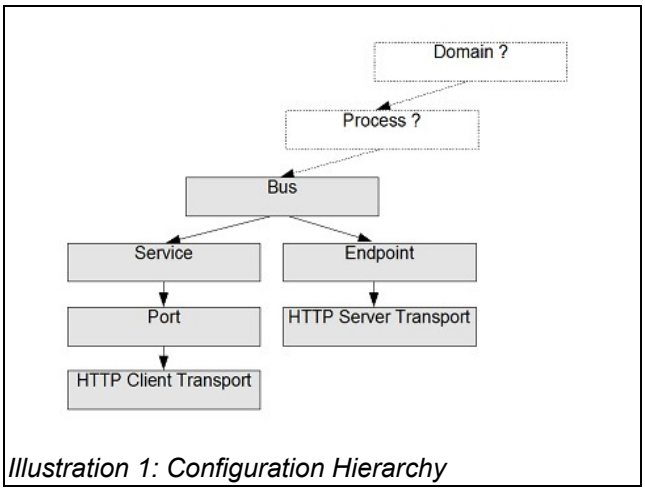

Illustration 1: Configuration Hierarchy

Later in this document, you will see how an understanding of this hierarchy is used to identify the component that a collection of configuration settings should apply to.

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. However, 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 1.0 General Availability release (April 2006); 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 Celtix Metadata XML 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 (...).

Beans element

In the opening `beans` element, you must 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 in Celtix.

Bean element

The example file in this section shows three ways of specifying a `bean` element. The first approach is used to define an abstract configuration, which is a configuration that is reused in another `bean` element. Because this declaration only describes configuration entries that are 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`.

Id attribute

The `id` attribute represents the component to which the configuration is 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 (for example, configuration entries for the Celtix routing service). The `class` attribute provides the class name of a bean in the Spring Framework infrastructure that is responsible for managing the configuration entries.

Property element

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.

Specifying 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 `property` element and the `value` elements, you need to use the types that are defined in several schema and metadata XML files, which are described in the next section.

The Celtix Metadata XML 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 in the `celtix.jar` file, located in the `CELTIX_HOME/lib` and `CELTIX_HOME/resources` directories.

When you are writing a Celtix configuration file, you must refer to the copies of these files that ship with your product to be certain that you have correctly specified 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, 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 content of each of the Celtix XML metadata files adheres to the structure defined in the following schema file:

```
resources/schemas/configuration/metadata.xsd
```

Each `<configItem>` element represents a configurable variable for the component.

The Celtix XML metadata files are as follows:

bus-config.xml

The `resources/config-metadata/bus-config.xml` file defines the metadata for configuring the bus component. This file describes the following configurable variables:

- `bindingFactories`
- `transportFactories`
- `resourceResolvers`.

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 the following namespace declaration: `org.objectweb.celtix.bus.bus_config.spring.BusConfigBean`.

endpoint-config.xml

The `resources/config-metadata/endpoint-config.xml` file defines the metadata for configuring the endpoint (server) component. This file describes the following configurable variables:

- `handlerChain`
- `systemHandlerChain`
- `serverContextInspectors`
- `enableSchemaValidation`

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`.

http-client-config.xml

The `resources/config-metadata/http-client-config.xml` file defines the metadata for configuring the service (client) HTTP transport. This file describes the following configurable variables:

- `httpClient`
- `authorization`
- `proxyAuthorization`
- `ssl`

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 http client 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`.

http-listener-config.xml

The `resources/config-metadata/http-listener-config.xml` file defines the metadata for configuring an HTTP transport listener. This file describes the following configurable variables:

- `httpListener`
- `ssl`

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 http listener 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`.

http-server-config.xml

The `resources/config-metadata/http-server-config.xml` file defines the metadata for configuring the endpoint (server) HTTP transport. This file describes the following configurable variables:

- `httpServer`
- `authorization`
- `ssl`

When writing a configuration file that includes http server 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`.

instrumentation-config.xml

The `resources/config-metadata/instrumentation-config.xml` file defines the metadata for configuring instrumentation. This file describes the following configurable variables:

- `InstrumentationControl`
- `MBServer`

When writing a configuration file that includes instrumentation related configuration entries, the value of the `class` attribute in the `bean` element is: `org.objectweb.celtix.bus.instrumentation.spring.InstrumentationConfigBean`.

jms-client-config.xml

The `resources/config-metadata/jms-client-config.xml` file defines the metadata for configuring the service (client) JMS transport. This file describes the following configurable variables:

- `jmsClient`
- `jmsAddress`

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 jms client 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`.

jms-server-config.xml

The file `resources/config-metadata/jms-server-config.xml` defines the metadata for configuring the endpoint (server) JMS transport. This file describes the following configurable variables:

`jmsServer`

`jmsAddress`

When writing a configuration file that includes jms server 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`.

port-config.xml

The `resources/config-metadata/port-config.xml` file defines the metadata for configuring the endpoint's port component. This file describes the following configurable variables:

- `address`
- `bindingId`
- `transportId`
- `handlerChain`
- `systemHandlerChain`
- `enableSchemaValidataion`.

The namespace assigned to this file's content is `http://celtix.objectweb.org/bus/jaxws/port-config`. When writing a configuration file that includes port related configuration entries, the value of the `class` attribute in the `bean` element is: `org.objectweb.celtix.bus.jaxws.port_config.spring.PortConfigBean`.

rm-config.xml

The `resources/config-metadata/rm-config.xml` file defines the metadata for configuring WS-Reliable Messaging (WS-RM). This file describes three configurable variables:

- `rmAssertion`

- `sourcePolicies`
- `destinationPolicies`.

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`.

router-config.xml

The `resources/config-metadata/router-config.xml` file defines the metadata for configuring the Celtix routing service. This file describes the `routesWSDL` configurable variable. 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`.

service-config.xml

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

The Celtix Schema Files

Your starting point for specifying a configuration entry is the metadata XML file. This enables you to 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 that you want to configure. Then refer 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 the following subdirectories:

- `resources/schemas/configuration`
- `resources/schemas/wsdl`

A simple example using the Celtix routing service will help to illustrate.

Configuring the Routing Service

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

```
<?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 the following information from this file (highlighted in the example XML):

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

Using this information, you can start 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 that you assign to the `bean` element's `id` attribute must be a unique string. Because 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 need to derive the `id` from the bus identifier. You can simply specify a string value instead.

To complete the `value` element, you must refer to the associated schema file. The `resources/schemas/wsdl/routing.xsd` schema file is defined in the `http://celtix.objectweb.org/routing/configuration` namespace. 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 in a WSDL file, and do not relate to configuring the routing service. The `wsdlUrl` and `urlListPolicy` types (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>
</xs:schema>
```

In the `router-config.xml` metadata XML file, the type for the `routesWSDL` configurable variable is `urlListPolicy`. This 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 `wsdlUrl` element wraps the `urlListPolicy` type. So to complete the configuration entry, you need to 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>
```

This simple configuration example uses the `wsdlUrl` and `url` elements, and not the `urlListPolicy` type . A more complex example is presented in the next section.

Writing a Celtix Configuration File

Review the structure of a Celtix configuration file (see page 2). You can 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. This 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. In 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

In the opening `beans` element, you should include namespace declarations corresponding to the Celtix schema files:

- `org/objectweb/celtix/configuration/config-metadata/types.xsd`
- `org/objectweb/celtix/configuration/config-metadata/metadata.xsd`.

These declarations are then available to all `bean` elements in 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

The section on The Celtix Metadata XML Files explains how the namespace used in the metadata XML file is mapped to the name of a Java bean class in the Spring Framework. This class name is 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 manages 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`

Component	Bean Class Name
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

The value of the `id` attribute can indicate what component the configuration applies to, however, it is not a simple one-to-one relationship like 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 that you need to specify the `id` are highlighted:

- the targetNamespace
- the service name
- 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`

Component	Instance Identifier	id Value
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

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

Setting the endpoint URL

The most straightforward example is where you want to configure the client application so that the URL used to invoke on the endpoint is defined in the configuration file instead of the WSDL file's content. The following fragment shows the corresponding `property` element for the hello world example. This corresponds to the configuration entry that sets 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>
```

Because 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. In the `value` element, 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

The server configuration file (`celtix-server.xml`) included in the `handlers` product demo shows a more complex example:

```
<?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>
```

Because 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. In the `value` element, you need to make entries that identify the class in your application that implements the handler you want to deploy.

Deciding what to place in the `value` element is a little more complex than in the previous example. In the metadata XML file, the `handlerChain` entry corresponds to the `handlerChainType` defined in the schema file `jaxws-types.xsd`. However, because 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>
```

Specifying initialization parameters

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 to the handler's initialization values. Because 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>
```

Specifying authorization information

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

In the metadata XML file, the `authorization` entry corresponds to the `AuthorizationPolicy` type defined in the schema file `security.xsd`. However, because 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 look up the composition of the `AuthorizationPolicy` type and determine that it is a sequence of the following 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.HttpCli
entConfigBean">
    <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 `configItem` element `httpClient` in the `http-client-config.xml metadata file` indicates that the `HTTPClientPolicy` type can be used to set transport attributes. The `HTTPClientPolicy` type, defined in the `resources/schemas/wsdl/http-conf.xsd` schema file, 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. 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.HttpCli
entConfigBean">
    <property name="httpClient">
      <value>
        <http-conf:client ProxyServer="localhost" ProxyServerPort="5049"
                          AutoRedirect="true" />
      </value>
    </property>
  </bean>
</beans>
```

An Example Application

Celtix 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 use the configuration file.

The Application Code

For this example, you build on the `hello_world` product demo. You need to perform the following steps:

1. Build this demo and confirm that it runs successfully using both the `ant` utility and the `java` executable, as described in the demo `README` file.

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

```
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, as follows:

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

2. 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).

3. Save the WSDL file.

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

4. 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

5. Write a configuration file that sets the address property of the `PortConfigBean`, for example:

```
<?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>
```

6. 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.HttpCli
entConfigBean">
    <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.

7. Save this file in text format into the *`installationDirectory`*`/celtix/samples/hello_world` directory; 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

8. 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.
9. 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.
10. 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
```

11. 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`.
12. 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>
```