Java™ API for XML Web Services 2.1 Change Log
March 16, 2007

Description
Maintenance revision of the Java API for XML Web Services, version 2.1. The main purpose of this change is to incorporate the WS-Addressing[33,34] functionality into JAX-WS, although some other minor additions will be proposed.

Maintenance Leads
Doug Kohlert, Sun Microsystems, Inc.
Arun Gupta, Sun Microsystems, Inc.

Feedback
Comments should be sent to jsr224-spec-comments@sun.com

Proposed changes
1 Introduction
1.1 Add the following at the end of the JAXB paragraphs
JAX-WS 2.1 requires JAXB 2.1 which is being developed in parallel with JAX-WS 2.1.

1.5 Add the following prefix definitions
Add the following prefixes to Table 1.1

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsa</td>
<td><a href="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing</a></td>
<td>The namespace for the WS-Addressing 1.0 schema [33]</td>
</tr>
</tbody>
</table>

2 WSDL 1.1 to Java Mapping

2.2 Add XmlSeeAlso requirements
A WSDL may define additional types via type substitution that are not referenced by a service directly but may still need to be marshalled by JAX-WS. The javax.xml.bind.XmlSeeAlso annotation from JAXB is used on the generated SEI to specify any additional types from the WSDL.

Conformance (javax.xml.bind.XmlSeeAlso required): An SEI generated from a WSDL that defines types not directly referenced by the Port MUST contain the
javax.xml.bind.XmlSeeAlso annotation with all of the additional types referenced either directly or indirectly.

Figure 2.1 shows how an SEI can be annotated with javax.xml.bind.XmlSeeAlso. This figure shows some of the types that may have been created while importing a WSDL and the different approaches to annotating the SEI.

```java
package example;
public class A { ... }

package example1;
public class B extends A { ... }

package example2;
public class C extends A { ... }

// Directly annotated SEI with classes B and C
@WebService
@XmlSeeAlso({B.class, C.class})
public interface MyService {
    public A echo(A a);
}

// Indirectly annotated SEI using generated JAXB // ObjectFactories
@XmlSeeAlso({example1.ObjectFactory.class,
              example2.ObjectFactory.class})
public interface MyService {
    public A echo(A a);
}
```

Figure 2.1 XmlSeeAlso annotation uses

### 2.3.1 Add the following to section 2.3.1

When generating an SEI from WSDL and XML schema, occasionally ambiguities occur on what XML infoset should be used to represent a method's return value or parameters. In order
to remove these ambiguities, JAXB annotations may need to be generated on methods and method parameters to assure that the return value and the parameters are marshalled with the proper XML infoset. A JAXB annotation on the method is used to specify the binding of a methods return type while an annotation on the parameter specifies the binding of that parameter. If the default XML infoset for the return type or parameters correctly represents the XML infoset, no JAXB annotations are needed.

Conformance (use of JAXB annotations): An SEI method MUST contain the appropriate JAXB annotations to assure that the proper XML infoset is used when marshalling/unmarshalling the return type. Parameters of an SEI method MUST contain the appropriate JAXB annotations to assure that the proper XML infoset is used when marshalling/unmarshalling the parameters of the method. The set of JAXB annotations that MUST be supported are: javax.xml.bind.annotation.XmlAttachmentRef, javax.xml.bind.annotation.XmlList, javax.xml.bind.XmlMimeType and javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter.

2.3.1.2 add “if present” to items (iii) and (iv)

Change the following:

(iii) The output message part refers to a global element declaration

(iv) The elements referred to by the input and output message parts (henceforth referred to as wrapper elements) are both complex types defined using the xsd:sequence compositor.

To:

(iii) The output message (if present) part refers to a global element declaration

(iv) The elements referred to by the input and output message (if present) parts (henceforth referred to as wrapper elements) are both complex types defined using the xsd:sequence compositor.

2.4 Change the first sentence of this section

Change:

“Mapping of XML Schema types to Java is described by the JAXB 2.0 specification[10].”

to:

“Mapping of XML Schema types to Java is described by the JAXB 2.1 specification[35].”

2.4.1 Add section 2.4.1 W3CEndpointReference

JAXB 2.1 by default does not map wsa:EndpointReference to the javax.xml.ws.wsaddressing.W3CEndpointReference class. However, for JAX-WS developers to fully utilize the use of a wsa:EndpointReference, JAX-WS implementations MUST map the wsa:EndpointReference to javax.xml.ws.wsaddressing.W3CEndpointReference. JAXB 2.1 provides a standard customization that can be used to force this mapping.

Conformance (javax.xml.ws.wsaddressing.W3CEndpointReference): Any schema element of
the type `wsa:EndpointReference` MUST be mapped to `javax.xml.ws.wsaddressing.W3CEndpointReference`.

### 2.7 Add description of new `getPortName(WebServiceFeature...)` method

Change the following sentence from:

“For each port in the service, the generated client side service class contains the following methods, one for each port defined by the WSDL service and whose binding is supported by the JAX-WS implementation:”

To:

“For each port in the service, the generated client side service class contains the following methods, two for each port defined by the WSDL service and whose binding is supported by the JAX-WS implementation:”

- **getPortName(WebServiceFeature... features)**
  One required method that takes a variable-length array of `javax.xml.ws.WebServiceFeature` and returns a proxy that implements the mapped service endpoint interface. The method generated delegates to the `Service.getPort(QName portName, Class<T> SEI, WebServiceFeature... features)` method passing it the port name, the SEI and the features. The value of the port name MUST be equal to the value specified in the mandatory `WebEndpoint` annotation on the method itself.

### 2.7 Change the following sentence

Change

“An application MAY customize the name of the generated method for a port using the `jaxws:method` binding declaration defined in section 8.7.8.”

to

“An application MAY customize the name of the generated methods for a port using the `jaxws:method` binding declaration defined in section 8.7.8.”

### 2.7.1 Fix the `getPortName()` samples so they take a QName for `portName`

The samples in the 2.0 specification was incorrectly passing just the local name of the `portName`, not the entire QName. This change only fixes the samples and does not change the APIs.

```java
@WebEndpoint(name="StockQuoteHTTPPort")
public StockQuoteProvider getStockQuoteHTTPPort() {
    return (StockQuoteProvider)super.getPort(
        new QName("http://example.com/stocks","StockQuoteHTTPPort"),
        stockQuoteProvider.class);
}
```

```java
@WebEndpoint(name="StockQuoteSMTPPort")
public StockQuoteProvider getStockQuoteSMTPPort() {
    return (StockQuoteProvider)super.getPort(
        new QName("http://example.com/stocks","StockQuoteSMTPPort"),
        stockQuoteProvider.class);
}
```
2.7.1 Add the `getPortName(WebServiceFeature...)` methods to the example

```java
@WebEndpoint(name="StockQuoteHTTPPort")
public StockQuoteProvider getStockQuoteHTTPPort(WebServiceFeature... features) {
    return (StockQuoteProvider)super.getPort(  
        new QName("http://example.com/stocks","StockQuoteHTTPPort"),  
        stockQuoteProvider.class,  
        features);
}

@WebEndpoint(name="StockQuoteSMTPPort")
public StockQuoteProvider getStockQuoteSMTPPort(WebServiceFeature... features) {
    return (StockQuoteProvider)super.getPort(  
        new QName("http://example.com/stocks","StockQuoteSMTPPort"),  
        StockQuoteProvider.class,  
        features);
}
```

3 Java to WSDL 1.1 Mapping

3.4 Add the following just prior to section 3.4.1

Multiple SEIs in the same package may result in name clashes as the result of sections 3.6.2.1 and 3.7 of the specification. Customizations may be used to resolve these clashes. See sections 7.2, 7.3 and 7.4 for more information on these customizations.

3.6 Add the following to section 3.6 Method Parameters and Return Type

Since JAX-WS uses JAXB for it data binding, JAXB annotations on methods and method parameters MUST be honored. A JAXB annotation on the method is used to specify the binding of a methods return type while an annotation on the parameter specifies the binding of that parameter.

**Conformance (use of JAXB annotations):** An implementation MUST honor any JAXB annotation that exists on an SEI method or parameter to assure that the proper XML infoset is used when marshalling/unmarshalling the the return value or parameters of the method. The set of JAXB annotations that MUST be supported are:

```java
javax.xml.bind.annotation.XmlAttachementRef,  
javax.xml.bind.annotation.XmlList, javax.xml.bind.XmlMimeType and
```
javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter.

3.6.2 Change the first sentence of this section
Change the sentence:
“JAXB defines a mapping from Java classes to XML Schema constructs.”
to:
“JAXB 2.1 defines a mapping from Java classes to XML Schema constructs.”

3.7 Add the following to section 3.7
Service specific exceptions are defined as all checked exceptions except
java.rmi.RemoteException and its subclasses.

Conformance (java.lang.RuntimeExceptions and java.rmi.RemoteExceptions)
java.lang.RuntimeException and java.rmi.RemoteException and their subclasses MUST NOT be treated as service specific exceptions and MUST NOT be mapped to WSDL.

4 Client APIs

4.2 Add getting of an EndpointReference
A web service client can get a javax.xml.ws.EndpointReference from a BindingProvider instance that will reference the target endpoint.

Conformance (Required BindingProvider getEndpointReference): An implementation MUST be able to return a javax.xml.ws.EndpointReference for the target endpoint if a SOAP binding is being used. If the BindingProvider instance has a binding that is either SOAP 1.1/HTTP or SOAP 1.2/HTTP, then a W3CEndpointReference MUST be returned. If the binding is XML/HTTP an java.lang.UnsupportedOperationException MUST be thrown.

4.2.3 Add additional getPort methods
Add the following getPort methods to the methods of a Service instance.

T getPort(Class<T> sei, WebServiceFeature... features) Returns a proxy for the specified SEI, the Service instance is responsible for selecting the port (protocol binding and and endpoint address). The specified features MUST be enabled/disabled and configured as specified.

T getPort(QName port, Class<T> sei, WebServiceFeature... features) Returns a proxy for the endpoint specified by port. Note that the namespace component of port is the target namespace of the WSDL definition document. The specified features MUST be enabled/disabled and configured as specified.
T getPort(EndpointReference epr, Class<T> sei, WebServiceFeature... features) Returns a proxy for the endpoint specified by epr. The address stored in the epr MUST be used during invocations on the endpoint. The endpointReference MUST NOT be used as the value of any addressing header such as wsa:ReplyTo. The specified features MUST be enabled/disabled and configured as specified. Any JAX-WS supported epr metadata MUST match the Service instance's ServiceName, otherwise a WebServiceException MUST be thrown. Any JAX-WS supported epr metadata MUST match the PortName for the sei, otherwise a WebServiceException MUST be thrown. If the Service instance has an associated WSDL, its WSDL MUST be used to determine any binding information, any WSDL in a JAX-WS supported epr metadata MUST be ignored. If the Service instance does not have a WSDL, then any WSDL inlined in the JAX-WS supported metadata of the epr MUST be used to determine binding information. If there is not enough metadata in the Service instance or in the epr metadata to determine a port, then a WebServiceException MUST be thrown.

4.3 Add the following just before section 4.3.1
A JAX-WS implementation MUST honor all WebServiceFeatures (section 6.5) for Dispatch based applications.

4.5 Add section “javax.xml.ws.EndpointReference”
An javax.xml.ws.EndpointReference is an abstraction that represents an invocable web service endpoint. Client applications can use an EndpointReference to get a port for an SEI although doing so prevents them from getting/setting the Executor or HandlerResolver which would normally be done on a Service instance. The EndpointReference class delegates to the javax.xml.ws.spi.Provider to perform the getPort operation. The following method can be used to get a proxy for a Port.

getPort(Class<T> serviceEndpointInterface, WebServiceFeature... features) Gets a proxy for the serviceEndpointInterface that can be used to invoke operations on the endpoint referred to by the EndpointReference instance. The specified features MUST be enabled/disabled and configured as specified. The returned proxy MUST use the EndpointReference instance to determine the endpoint address and any reference parameters to be sent on endpoint invocations. The EndpointReference instance MUST NOT be used directly as the value of an WS-Addressing header such as wsa:ReplyTo. For this method to successfully return a proxy, WSDL metadata MUST be available and the EndpointReference instance MUST contain an implementation understood serviceName in its metadata.

5 Service APIs
5.1 Add the following just before section 5.1.1
A JAX-WS implementation MUST honor all WebServiceFeatures (section 6.5) for Provider based applications.

5.2.8 Add new section javax.xml.ws.EndpointReference
The following methods can be used on a published Endpoint to retrieve an javax.xml.ws.EndpointReference for the Endpoint instance.

getEndpointReference(List<Element> referenceParameters) Creates and returns and javax.xml.ws.EndpointReference for a published Endpoint. If the binding is SOAP 1.1/HTTP or SOAP 1.2/HTTP, then a javax.xml.ws.wsaddressing.W3CEndpointReference MUST be returned. A returned W3CEndpointReference MUST also contain the specified referenceParameters. An implementation MUST throw a javax.xml.ws.WebServiceException if the Endpoint instance has not been published. An implementation MUST throw java.lang.UnsupportedOperationException if the Endpoint instance uses the XML/HTTP binding.

getEndpointReference(Class<T> clazz, List<Element> referenceParameters) Creates and returns and javax.xml.ws.EndpointReference of type clazz for a published Endpoint instance. If clazz is of type javax.xml.ws.wsaddressing.W3CEndpointReference, then the returned W3CEndpointReference MUST contain the specified referenceParameters. An implementation MUST throw a javax.xml.ws.WebServiceException if the Endpoint instance has not been published. If the Class clazz is not a subclass of EndpointReference or the Endpoint implementation does not support EndpointReferences of type clazz a javax.xml.ws.WebServiceException MUST be thrown. An implementation MUST throw java.lang.UnsupportedOperationException if the Endpoint instance uses the XML/HTTP binding.

5.4 add section “W3CEndpointReferenceBuilder”
Occasionally it is necessary for one application component to create an EndpointReference for another web service endpoint. The W3CEndpointReferenceBuilder class provides a standard API for creating W3CEndpointReferences for web service endpoints.

6 Core APIs

6.2.2 Amend the description of the createEndpoint method
Change:
createEndpoint(String bindingId, Object implementor) Creates and returns an Endpoint for the specified binding and implementor.
To:
createEndpoint(String bindingId, Object implementor) Creates and returns an Endpoint for the specified binding and implementor. If the bindingId is null and no binding information is specified via the javax.xml.ws.BindingType annotation then a default
SOAP1.1/HTTP binding MUST be used.

6.2.4 Add a section “Creating EndpointReferences”
The Provider class provides the following methods to create EndpointReference instances.

`readEndpointReference(javax.xml.transform.Source source)` Unmarshalls and returns a `javax.xml.ws.EndpointReference` from the infoset contained in source.

`createW3CEndpointReference` Creates a `W3CEndpointReference` using the specified String address, QName serviceName, QName portName, List<Element> metadata, String wsdlDocumentLocation, and List<Element> referenceParameters parameters.

6.2.5 Add a section “Getting Port Objects”
The following method can be used to get a proxy for a Port.

`getPort(EndpointReference epr, Class<T> sei, WebServiceFeature... features)` Gets a proxy for the sei that can be used to invoke operations on the endpoint referred to by the epr. The specified features MUST be enabled/disabled and configured as specified. The returned proxy MUST use the epr to determine the endpoint address and any reference parameters that MUST be sent on endpoint invocations. The epr MUST NOT be used directly as the value of an WS-Addressing header such as wsa:ReplyTo.

6.5 Add Section 6.5 `javax.xml.ws.WebServiceFeature`
JAX-WS 2.1 introduces the notion of features. A feature is associated with a particular functionality or behavior. Some features may only have meaning when used with certain bindings while other features may be generally useful. JAX-WS 2.1 introduces three standard features, AddressingFeature, MTOMFeature and RespectBindingFeature as well as the base WebServiceFeature class. A JAX-WS 2.1 implementation may define its own features but they will be non-portable across all JAX-WS 2.1 implementations.

Each feature is derived from the `javax.xml.ws.WebServiceFeature` class. This allows the web service developer to pass different types of `WebServiceFeatures` to the various JAX-WS APIs that utilize them. Also, each feature should be documented using JavaDocs on the derived classes. Each `WebServiceFeature` MUST have a public static final String ID field that is used to uniquely identify the feature.

*Conformance (javax.xml.ws.WebServiceFeatures):* Each derived type of `javax.xml.ws.WebServiceFeature` MUST contain a public static final String ID field that uniquely identifies the feature against all features of all implementations.
Since vendors can specify their own features, care MUST be taken when creating a feature ID so as to not conflict with another vendor's ID.

The WebServiceFeature class also has an enabled property that is used to store whether a particular feature should be enabled or disabled. Each derived type should provide either a constructor argument and/or a method that will allow the web service developer to set the enabled property. The meaning of enabled or disabled is determined by each individual WebServiceFeature. It is important that web services developers be able to enable/disable specific features when writing their web applications. For example, a developer may choose to implement WS-Addressing himself while using the Dispatch and Provider APIs and thus he MUST be able to tell JAX-WS to disable addressing.

Conformance (enabled property): Each derived type of javax.xml.ws.WebServiceFeature MUST provide a constructor argument and/or method to allow the web service developer to set the value of the enabled property. The public default constructor MUST by default set the enabled property to true. An implementation MUST honor the value of the enabled property of any supported WebServiceFeature.

6.5.1 Add Section 6.5.1 javax.xml.ws.soap.AddressingFeature
The AddressingFeature is used to control the use of WS-Addressing[33] by JAX-WS. This feature MUST be supported with the SOAP 1.1/HTTP or SOAP 1.2/HTTP bindings. Using this feature with any other binding is undefined. This feature corresponds to the Addressing annotation described in section 7.14.1.

Enabling this feature on the server will result in the runtime being capable of consuming and responding to WS-Addressing headers.

Enabling this feature on the client will cause the JAX-WS runtime to include WS-Addressing headers in SOAP messages as specified by WS-Addressing[33].

Disabling this feature will prevent a JAX-WS runtime from processing or adding WS-Addressing headers from/to SOAP messages even if the associated WSDL specifies otherwise. This may be necessary if a client or endpoint needs to implement Addressing themselves. For example, a client that desires to use non-anonymous ReplyTo can do so by disabling the AddressingFeature and by using Dispatch<Source> with Message mode.

The AddressingFeature has one property required, that can be configured to control whether all incoming messages MUST contain Addressing headers.

The AddressingFeature MAY be automatically enabled if the WSDL specifies its use in a manner supported by an implementation. Developers may choose to prevent this from happening by explicitly disabling the AddressingFeature.

6.5.1.1 Add section 6.5.1.1 javax.xml.ws.EndpointReference
The abstract EndpointReference class is used by the JAX-WS APIs to reference a
particular endpoint in accordance with the W3C Web Services Addressing 1.0 [33]. Each concrete instance of an EndpointReference MUST contain a wsa:Address.

Applications may also use the javax.xml.ws.EndpointReference class in method signatures. JAXB 2.1 will will bind the EndpointReference base class to xs:anyType. Applications should instead use concrete implementations of EndpointReference such as javax.xml.ws.W3CEndpointReference which will provide better binding. JAX-WS implementations are required to support the W3CEndpointReference class but they may also provide other EndpointReference subclasses that represent different versions of Addressing.

6.5.1.2 Add Section 6.5.1.2 javax.xml.ws.W3CEndpointReference
The W3CEndpointReference class is a concrete implementation of the javax.xml.ws.EndpointReference class and is used to reference endpoints that are compliant with the W3C Web Services Addressing 1.0 – Core [33]recommendation. Applications may use this class to pass EndpointReference instances as method parameters or return types. JAXB 2.1 will bind the W3CEndpointReference class to the W3C EndpointReference XML Schema in the WSDL.

6.5.2 Add Section javax.xml.ws.soap.MTOMFeature
The MTOMFeature is used to specify if MTOM should be used with a web service. This feature should be used instead of the javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING, javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_MTOM_BINDING and the javax.xml.ws.soap.SOAPBinding.setMTOMEnabled(). This feature MUST be supported with the SOAP 1.1/HTTP or SOAP 1.2/HTTP bindings. Using this feature with any other bindings is undefined. This feature corresponds to the MTOM annotation described in section 7.14.2.

Enabling this feature on either the server or client will result the JAX-WS runtime using MTOM and binary data being sent as an attachment.

The MTOMFeature has one property threshold, that can be configured to serve as a hint for which binary data SHOULD be sent as an attachment. The threshold is the size in bytes that binary data SHOULD be in order to be sent as an attachment. The threshold MUST not be negative. The default value is 0.

Conformance (javax.xml.ws.soap.MTOMFeature): An implementation MUST support the javax.xml.ws.soap.MTOMFeature and its threshold property.

6.5.3 Add Section javax.xml.ws.RespectBindingFeature
The RespectBindingFeature is used to control whether a JAX-WS implementation MUST respect/honor the contents of the wsdl:binding associated with an endpoint. It has a corresponding RespectBinding annotation described in section 7.14.3.

Conformance (javax.xml.ws.RespectBindingFeature): When the
javax.xml.ws.RespectBindingFeature is enabled, a JAX-WS implementation MUST inspect the wsdl:binding at runtime to determine result and parameter bindings as well as any wsdl:extensions that have the required="true" attribute. All required wsdl:extensions MUST be supported and honored by a JAX-WS implementation unless a specific wsdl:extension has be explicitly disabled via a WebServiceFeature.

In order to not break backward compatibility with JAX-WS 2.0, the behavior with regards to respecting the wsdl:binding when this feature is disabled is undefined.

7 Annotations

Add the following conformance requirement to section 7.

Conformance (Unsupported WebServiceFeatureAnnotations): If an unrecognized or unsupported annotation annotated with the WebServiceFeatureAnnotation meta-annotation:

- In a client setting, an implementation MUST NOT invoke any remote operations, if any. Instead, it MUST throw a WebServiceException, setting the cause to an exception approximating the cause of the error (e.g. an IllegalArgumentException or a ClassNotFoundException).
- In a server setting, annotation, an implementation MUST NOT dispatch to an endpoint implementation object. Rather it MUST generate a fault appropriate to the binding in use.

7.12 Add javax.xml.ws.Action

The Action annotation is applied to the methods of a SEI. It is used to specify the input, output WS-Addressing Action values associated with the annotated method.

In this version of JAX-WS there is no standard way to specify Action values in a WSDL and there is no standard default value. It is intended that, after the W3C WG on WS-Addressing has defined these items in a recommendation, a future version of JAX-WS will require the new standards.

Table 7:11

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault</td>
<td>Array of FaultAction for {} the wsdl:Faults of the operation</td>
<td>{}</td>
</tr>
<tr>
<td>input</td>
<td>Action for the wsdl:input &quot;&quot; of the operation</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>output</td>
<td>Action for the wsdl:output of the operation</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

7.13 Add javax.xml.ws.FaultAction
The FaultAction annotation is used within the Action annotation to specify the WS-Addressing Action of a service specific exception as defined by section 3.7.

In this version of JAX-WS there is no standard way to specify Action values in a WSDL and there is no standard default value. It is intended that, after the W3C WG on WS-Addressing has defined these items in a recommendation, a future version of JAX-WS will require the new standards.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Action for the wsdl:fault&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>className</td>
<td>Name of the exception class</td>
<td>No defaults, required property</td>
</tr>
</tbody>
</table>

**7.14 Add section javax.xml.ws.spi.WebServiceFeatureAnnotation**

The WebServiceFeatureAnnotation is a meta-annotation used by a JAX-WS implementation to identify other annotations as WebServiceFeatures. JAX-WS provides the following annotations as WebServiceFeatures: javax.xml.ws.soap.Addressing, javax.xml.ws.soap.MTOM, and javax.xml.ws.RespectBinding. If a JAX-WS implementation encounters an annotation annotated with the WebServiceFeatureAnnotation that it does not support or recognize an ERROR MUST be given.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identifier for the WebServiceFeature</td>
<td>No defaults required property</td>
</tr>
<tr>
<td></td>
<td>represented by the annotated annotation.</td>
<td></td>
</tr>
<tr>
<td>bean</td>
<td>The class name of a derived WebServiceFeature</td>
<td>No defaults required property</td>
</tr>
<tr>
<td></td>
<td>class associated with the annotated annotation.</td>
<td></td>
</tr>
</tbody>
</table>

The following shows how the Addressing annotation uses the WebServiceFeatureAnnotation meta-annotation.

```java
@WebServiceFeatureAnnotation(id=AddressingFeature.ID,
                            bean=AddressingFeature.class)
public @interface Addressing {
    /**
     * Specifies if this feature is enabled or disabled.
     */
    boolean enabled() default true;
```
/ **
 *
 * Property to determine if WS-Addressing headers
 * MUST be present on incoming messages.
 * /
  boolean required() default false;

7.14.1 Added section javax.xml.ws.soap.Addressing
The Addressing annotation is applied to an endpoint implementation class. It is used to
control the use of WS-Addressing\[33\][34]. It corresponds with the AddressingFeature
described in section 6.5.1.

Table 7:14

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Specifies if WS-Addressing is enabled or not</td>
<td>true</td>
</tr>
<tr>
<td>required</td>
<td>Specifies whether Addressing headers MUST be present on incoming messages.</td>
<td>false</td>
</tr>
</tbody>
</table>

The definition of this annotation is incomplete in this release of JAX-WS as there is no standard
way to convey the use of WS-Addressing via a WSDL and there is no standard definition for
the default value of WS-Addressing Action headers; however, the runtime behavior of this
annotation is well-defined. It is intended that a future version of JAX-WS will require the use of
the standard mechanism to convey the use of WS-Addressing via WSDL and default values for
WS-Addressing Action headers as defined by the W3C WG on WS-Addressing.

To write a portable endpoint and its corresponding client with this version of JAX-WS, an
endpoint MUST explicitly specify what WS-Addressing Actions are to be used via the Action
and FaultAction annotations. The client MUST explicitly enable addressing via the
AddressingFeature, and for each invocation, the client MUST explicitly set the
BindingProvider.SOAPACTION_URI_PROPERTY. After the W3C WG on WS-
Addressing has specified how the use of WS-Addressing is specified in the WSDL, and what
the default value must be for Action headers, a future version of JAX-WS will remove these
requirements.

7.14.2 Add section javax.xml.ws.soap.MTOM
The MTOM annotation is applied to an endpoint implementation class. It is used to control the
use of MTOM. It corresponds to the MTOMFeature described in section 6.5.2.

Table 7:15

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Specifies if MTOM is</td>
<td>true</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>threshold</td>
<td>Specifies the size in bytes that binary data SHOULD be before being sent as an attachment.</td>
<td>0</td>
</tr>
</tbody>
</table>

7.14.3 Add section `javax.xml.ws.RespectBinding`

The `RespectBinding` annotation is applied to an endpoint implementation class. It is used to control whether a JAX-WS implementation MUST respect/honor the contents of the `wsdl:binding` associated with an endpoint. It has a corresponding `RespectBindingFeature` described in section 6.5.3.

Table 7:16

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Specifies whether the <code>wsdl:binding</code> must be respected or not.</td>
<td>true</td>
</tr>
</tbody>
</table>

9 Handler Framework

9.4.1.1 Standard Message Context Properties

Add the `javax.xml.ws.reference.parameters` property to Table 9.2.

Name | Type | Mandatory | Description |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.xml.ws.reference.parameters</code></td>
<td>List&lt;Element&gt;</td>
<td>Y</td>
<td>A list of WS Addressing reference parameters. The list MUST include all SOAP headers marked with the <code>wsa:IsReferenceParameter=&quot;true&quot;</code> attribute.</td>
</tr>
</tbody>
</table>

10 SOAP Binding

10.4.1.5 Add Addressing section

If the `javax.xml.ws.soap.AddressingFeature` is enabled, implementations are required to follow WS-Addressing[33,34] protocols.

*Conformance (SOAP Addressing Support)*: An implementation MUST support WS-Addressing 1.0 – SOAP Binding[34].

Bibliography – add the following references
Accepted Changes

(Changes accepted by the EC will be moved to this section.)

Deferred Changes

(Changes deferred by the EC will be moved to this section.)