

Java™ API for XML Web Services 2.1 Change Log

October 176, 2006

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[32,33,34] 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

Prefix	Namespace	Notes
wsa	http://www.w3.org/2005/08/addressing	The namespace for the WS-Addressing 1.0 schema [33]
wsaw	http://www.w3.org/2006/05/addressing/wsdl	The namespace for the WS-Addressing 1.0 – WSDL Binding schema[32]

2 WSDL 1.1 to Java Mapping

Add the following conformance requirement to Chapter 2

Conformance (WSDL Addressing Support): An implementation MUST support the mapping of WS-Addressing 1.0 – WSDL Binding[32] to Java.

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.

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

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

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

```
@WebService
public interface MyService {
    public A echo(A 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 Add mapping of `wsa:Action` and `wsa:FaultAction`

Methods generated from `wsdl:input` and `wsdl:output` messages that contain a `wsaw:Action` attributes MUST be annotated with `javax.xml.ws.Action`. See section 7 for more information on these annotations.

Conformance (javax.xml.ws.Action): A mapped Java method MUST be annotated with a `javax.xml.ws.Action` annotation if the `wsdl:input` or `wsdl:output` elements contain a `wsaw:Action` attribute. If the `wsdl:input` element contains a `wsaw:Action`, the value of this attribute MUST be set to the `javax.xml.ws.Action.input` element. If the `wsdl:output` element contains a `wsaw:Action`, the value of this attribute MUST be set to the `javax.xml.ws.Action.output` element.

Methods generated from `wsdl:fault` messages that contain a `wsaw:Action` attributes MUST ~~must~~ be annotated with `javax.xml.ws.FaultAction`. See section 7 for more information on this annotation.

Conformance (javax.xml.ws.FaultAction): A mapped Java method MUST be annotated with a `javax.xml.ws.FaultAction` annotation if the `wsdl:fault` elements contain a `wsaw:Action` attribute. The `javax.xml.ws.FaultAction.value` is taken directly from the value of the `wsaw:Action`. The `javax.xml.ws.FaultAction.className` MUST be the exception class name associated with this `wsdl:fault`.

Figure 2.2 shows the mapping of a `wsdl:operation` containing input, output, and fault elements with `wsaw:Action` attributes.

```

<operation name="addNumbers">
  <input message="tns:add" wsaw:Action="inAction"/>
  <output message="tns:addResponse" wsaw:Action="outAction"/>
  <fault name="addFault" message="tns:addFault"
wsaw:Action="faultAction"/>
</operation>

// the mapped Java method will be

@Action(input="inAction",
        output="outAction",
        fault= {
    @FaultAction(className=AddFaultException.class,
                  value="faultAction")
        })
public int addNumbers(int number1, int number2) throws
AddFaultException;

```

Figure 2.2 Mapping of `wsaw:Action`

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 info set 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 info set. A JAXB annotation on the method is used to specify the binding of a method's return type while an annotation on the parameter specifies the binding of that parameter. If the default XML info set for the return type or parameters correctly represents the XML info set, 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 info set 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 info set is used when marshalling/unmarshalling the parameters of the method.

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 `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(WebServiceFeatures...)` 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(WebServiceFeatures... features)` One required method that takes a variable-length array of `javax.xml.ws.WebServiceFeature`s 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.

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

```

@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(WebServiceFeatures...)` methods to the example

```

@WebEndpoint(name="StockQuoteHTTPPort")
public StockQuoteProvider getStockQuoteHTTPPort(WebServiceFeatures...
                                                features) {

    return (StockQuoteProvider)super.getPort(
        new QName("http://example.com/stocks", "StockQuoteHTTPPort"),
        stockQuoteProvider.class,
        features);
}

```

```

@WebEndpoint(name="StockQuoteSMTPPort")
public StockQuoteProvider getStockQuoteSMTPPort(WebServiceFeatures...
                                                features) {

    return (StockQuoteProvider)super.getPort(
        new QName("http://example.com/stocks", "StockQuoteSMTPPort"),
        StockQuoteProvider.class,
        features);
}

```

3 Java to WSDL 1.1 Mapping

Add the following conformance requirement to Chapter 3

Conformance (WSDL Addressing Support): An implementation MUST support the mapping of Java to WS-Addressing 1.0 – WSDL Binding[32].

3.3 Add mapping of `javax.xml.ws.soap.Addressing` ~~annotation~~ **Feature**

Classes A service endpoint implementation class annotated with the ~~`javax.xml.ws.BindingType` annotation with the~~ `javax.xml.ws.soap.Addressing` ~~Feature~~ **annotation and with the `enabled` element set to `true`**, MUST result in the `wsaw:UsingAddressing` extensibility element on the `wsdl:binding`.

*Conformance (`javax.xml.ws.soap.Addressing` **Feature**):* A **service endpoint implementation Java** class that is annotated with the ~~`javax.xml.ws.BindingType` annotation that contains the~~ `javax.xml.ws.soap.Addressing` ~~Feature~~ **annotation** with the `enabled` element set to `true`, MUST result in the addition of a `wsaw:UsingAddressing` extensibility element to the `wsdl:binding` element and it MUST NOT have the `wsdl:required="true"` attribute. If the required **element** ~~`FeatureParameter` of the `Addressing` ~~Feature~~ annotation~~ has a value of `true`, **then the** `wsaw:UsingAddressing` **extensibility element MUST contain the** `wsdl:required="true"` **attribute**. If **the endpoint implementation class is also annotated with the a** `BindingType`.value **that** is not compatible with this feature an error MUST be given. The JAX-WS runtime MUST also use Addressing

headers. If the enabled element is set to false, then wsaw:UsingAddressing element MUST NOT be generated and the JAX-WS runtime MUST NOT use Addressing headers.

Figure 3.1 shows the mapping ~~of a BindingType annotation with the the~~ AddressingFeature ~~annotation enabled.~~

```
@BindingType(SOAPBinding.SOAP11HTTP_BINDING,  
features={@Feature(AddressingFeature.ID)})
```

```
@Addressing
```

```
public class AddNumbersImpl {  
    ...  
}
```

```
<definitions  
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"  
  ...>  
  ...  
<wsdl:binding ...>  
  <wsaw:UsingAddressing wsdl:required="false"/>  
  ...  
</wsdl:definitions>
```

Figure 3.1 Mapping of AddressingFeature

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.

3.5 Add mapping of javax.xml.ws.Action and javax.xml.ws.FaultAction

Conformance (javax.xml.ws.Action): A Java method annotated with the javax.xml.ws.Action.input annotation element MUST result in the addition of a wsaw:Action extensibility element to the wsdl:input element with the wsaw:Action.value equal to javax.xml.ws.Action.input. A Java method annotated with the javax.xml.ws.Action.output annotation element MUST result in the addition of a wsaw:Action extensibility element on the wsdl:output element with the wsaw:Action.value equal to javax.xml.ws.Action.output.

Conformance (javax.xml.ws.FaultAction) A Java method annotated with the javax.xml.ws.FaultAction annotation element MUST result in the addition of a wsaw:Action extensibility element on the wsdl:fault element that corresponds to the

Exception specified by `javax.xml.ws.FaultAction.className` with the `wsaw:Action.value` equal to `javax.xml.ws.FaultAction.value`.

Figure 3.2 shows the mapping of `javax.xml.ws.Action` and `javax.xml.ws.FaultAction` to a `wsdl:operation`.

```
@Action(input="inAction",
        output="outAction",
        fault={
            @FaultAction(className=AddNumbersException.class,
                          value="faultAction")
        })
public int addNumbersFault(int number1, int number2)
    throws AddNumbersException

<operation name="addNumbersFault">
<input wsaw:Action="inAction" message="tns:addNumbersFault"/>
<output wsaw:Action="outAction"
        message="tns:addNumbersFaultResponse"/>
<fault name="AddNumbersException"
        message="tns:AddNumbersException"
        wsaw:Action="faultAction"/>
</operation>
```

Figure 3.2 Mapping of Action and FaultAction to wsdl:operation

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

Since JAX-WS uses JAXB for its 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 method's 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 return value or parameters of the method.

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 an `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 an `javax.xml.ws.EndpointReference` for the target endpoint if a SOAP binding is being used. If the `BindingProvider` instance has ~~an associated WSDL,~~ ~~and the a~~ binding ~~that~~ is either SOAP 1.1/HTTP or SOAP 1.2/HTTP, then ~~a~~ W3CEndpointReference MUST be returned with the `wsaw:ServiceName` element with the `wsaw:EndpointName` attribute in the `wsa:Metadata`. If there is an associated the WSDL, ~~then the WSDL SHOULD~~ ~~MUST~~ be in-lined in the `wsa:Metadata`, ~~and the~~ ~~`wsa:Metadata` MUST also contain the `wsaw:ServiceName` element with the `wsaw:EndpointName` attribute.~~ **The `wsaw:InterfaceName` MAY be present in the `wsa:Metadata`.** 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, WebServiceFeatures... features)` Returns a proxy for the specified ~~sei~~ 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, WebServiceFeatures... 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, WebServiceFeatures... 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. The `epr`'s `wsaw:ServiceName` MUST match the `Service` instance's `ServiceName`, otherwise a `WebServiceException` MUST be

thrown. The `epr's wsaw:EndpointName` 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 the `epr` will be ignored. If the `Service` instance does not have a WSDL, then any WSDL inlined in the `epr` will be used to determine binding information.

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`.

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 an `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. If the `Endpoint` instance has an associated WSDL, then a returned `W3CEndpointReference` MUST in-line the WSDL in the `wsa:Metadata` and the `wsa:Metadata` MUST also contain the `wsaw:ServiceName` element with the `wsaw:EndpointName` attribute. 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` and if the `Endpoint` instance has an associated WSDL, then the WSDL MUST be in-lined in the `wsa:Metadata` and the `wsa:Metadata` MUST also contain the `wsaw:ServiceName` element with the `wsaw:EndpointName` attribute. 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. 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 “Reading-Creating EndpointReferences”

The `Provider` class provides the following methods to create `EndpointReference` instances.

~~javax.xml.ws.EndpointReferences can be created using the following method:~~

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. ~~This ID can be specified in the new `Feature` (see 7.8.1) annotation that is used in conjunction with the `BindingType` (see 7.8) annotation to enable or disable a 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 `wsaw:UsingAddressing` element being added to the `wsdl:Binding` in the generated WSDL if the WSDL does not already exist for the endpoint and 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 had the `wsaw:UsingAddressing` element with the `required="true"` attribute. 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 (~~`FeatureParameter`~~) required, that can be configured to control whether the generated `wsaw:UsingAddressing` element will contain the `required="true"` attribute.

The `AddressingFeature` can be automatically enabled if the `wsaw:UsingAddressing` extensibility element is in the `wsdl:binding`. 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 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 [33] and WS-Addressing – WSDL Binding[32]. When creating an `W3CEndpointReference`, it SHOULD contain the `wsaw:ServiceName` element with the `wsaw:EndpointName` attribute in the `wsa:Metadata`. If an associated WSDL is available, then the `W3CEndpointReference` ~~SHOULD~~**MUST** contain the WSDL inlined in the `wsa:Metadata`. ~~It MUST also contain the `wsaw:ServiceName` element with the `wsaw:EndpointName` attribute.~~ The `wsaw:InterfaceName` MAY be present. Applications may use this class to pass `EndpointReference` ~~s~~ 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 ~~any~~ binary data being sent as an attachment ~~XOP-encoded~~.

The `MTOMFeature` has one property (~~`FeatureParameter`~~) `threshold`, that can be configured to ~~control~~ serve as a hint for which binary data ~~SHOULD~~should be sent as an attachment ~~XOP-encoded~~. The threshold is the size in bytes that binary data ~~SHOULD~~**MUST** be in order to be sent as an attachment ~~XOP-encoded~~. 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 been explicitly disabled via a `WebServiceFeature`. For example, if a `wsdl:binding` has a `wsaw:UsingAddressing` element with the `required="true"` attribute, WS-Addressing MUST be enabled and used unless the web service developer explicitly disables the `WSAddressingFeature`.

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.

-

7.8 Add features to `BindingType` annotation

Add the following property to Table 7.8

Property	Description	Default
features	Features to be enabled on the specified binding	{}

7.8.1 Add `javax.xml.ws.Feature` annotation

~~Features are additional functionality that can be enabled on a web service. Each feature defines its purpose and default behavior and can define any number of feature parameters. This annotation allows the web service developer to enable features and to configure them via `FeatureParameters`.~~

Add the following Table 7.8.1

Property	Description	Default
value	The unique identifier for the feature	""

enabled	Specifies whether the feature is enabled or not	true
parameters	An array of parameters for this feature.	{}
Each feature defines which parameters it has		

7.8.1.1 Add javax.xml.ws.FeatureParameter annotation

The ~~FeatureParameter~~ annotation is used to specify the values for a feature parameter. Each feature defines any number of feature parameters.

Add the following Table 7.8.1.1

Property	Description	Default
name	The name of the parameter	""
value	The value of the parameter	""

The following is an example use of the BindingType annotation that enables WS-Addressing and MTOM and sets the MTOM threshold value to 1000.

```
@BindingType(SOAPBinding.SOAP11HTTP_BINDING,
    features={
        @Feature(AddressingFeature.ID),
        @Feature(MTOMFeature.ID,
            parameters={@FeatureParameter(
                name=MTOMFeature.THRESHOLD, value="1000")
            })
    })
public class MyWebService { ... }
```

7.12 Add javax.xml.ws.Action

The Action annotation is applied to the methods of a SEI. It is used to generate the wsa:Action on wsdl:input and wsdl:output of each wsdl:operation mapped from the annotated methods.

Table 7:112

Property	Description	Default
input	Action for the wsdl:input operation	""
output	Action for the wsdl:output operation	""

7.13 Add javax.xml.ws.FaultAction

The FaultAction annotation is used within the Action annotation to generate the wsa:Action element on the wsdl:fault element of each wsdl:operation mapped from the annotated methods.

Table 7:123

Property	Description	Default
value	Action for the wsdl:fault operation	""
className	Name of the exception class	No defaults, required property

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 7:13

Property	Description	Default
<u>id</u>	<u>Unique identifier for the <code>WebServiceFeature</code> represented by the annotated annotation.</u>	<u>No defaults required property</u>
<u>bean</u>	<u>The class name of a derived <code>WebServiceFeature</code> class associated with the annotated annotation.</u>	<u>No defaults required property</u>

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

```
@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 the value of the
     * <code>wsdl:required</code> attribute on
     * <code>wsaw:UsingAddressing</code> element in the WSDL.
     */
    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][35]. It corresponds with the AddressingFeature described in section 6.5.1.

Table 7:14

<u>Property</u>	<u>Description</u>	<u>Default</u>
<u>enabled</u>	<u>Specifies if WS-Addressing is enabled or not</u>	<u>true</u>
<u>required</u>	<u>Specifies the value of the wsdl:required attribute on the wsaw:UsingAddressing extensibility element in the wsdl:binding.</u>	<u>false</u>

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

<u>Property</u>	<u>Description</u>	<u>Default</u>
<u>enabled</u>	<u>Specifies if MTOM is enabled or not.</u>	<u>true</u>
<u>threshold</u>	<u>Specifies the size in bytes that binary data SHOULD be before being sent as an attachment.</u>	<u>0</u>

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

<u>Property</u>	<u>Description</u>	<u>Default</u>
<u>enabled</u>	<u>Specifies whether the wsdl:binding must be respected or not.</u>	<u>true</u>

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
<code>javax.xml.ws.reference.parameters</code>	List<Element>	Y	A list of WS Addressing reference parameters. The list MUST include all SOAP headers marked with the <code>wsa:IsReferenceParameter="true"</code> attribute.

10 SOAP Binding

10.4.1.2 Add Addressing section

If the `javax.xml.ws.soap.AddressingFeature` is enabled, implementations are required to follow WS-Addressing[32,33,34] protocols.

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

Bibliography – add the following references

[32] Martin Gudgin, Marc Hadley, Tony Rogers, Ümit Yalçinalp . Web Services Addressing 1.0 - WSDL Binding. W3C Candidate Recommendation 29 May 2006. See <http://www.w3.org/TR/2006/CR-ws-addr-wsdl-20060529/>.

[33] Martin Gudgin, Marc Hadley, Tony Rogers. Web Services Addressing 1.0 - Core. W3C Recommendation 9 May 2006. See <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/>.

[34] Martin Gudgin, Marc Hadley, Tony Rogers. Web Services Addressing 1.0 - SOAP Binding. W3C Recommendation 9 May 2006. See <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>.

[35] Kohsuke Kawaguchi. The Java Architecture of XML Binding (JAXB) 2.1. JSR, JCP August 2003. See <http://jcp.org/en/jsr/detail?id=222>.

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.)