# JSR110 JWSDL 1.2 Maintenance Release Change Log

This change log proposes changes to the JWSDL API to be delivered via a Maintenance Release of JSR110 "Java APIs for WSDL", as per the JCP process 2.1. Comments and feedback during the 30 day change log review period should be sent to the JSR110 mailing list jsr110-eg-disc@yahoogroups.com.

The API defined by the current release of JSR110 is JWSDL 1.1, with WSDL4J 1.5 as its reference implementation. This was established by the initial JSR110 maintenance release in January 2005. The URL for this JSR is http://jcp.org/en/jsr/detail?id=110 . WSDL4J is hosted by SourceForge.net at http://sourceforge.net/projects/wsdl4j/.

This proposed maintenance release will move the API to JWSDL 1.2, with the reference implementation provided by WSDL4J 1.6. During the 30 day change log review period, the proposed changes for JWSDL 1.2 and WSDL4J 1.6 will be made available in the WSDL4J CVS repository at SourceForge.net in a branch indicating WSDL4J 1.6. At completion of the maintenance release this branch will be merged to the CVS HEAD.

This change log captures enhancements to JWSDL and WSDL4J or issues raised by the WSDL4J user community via the mailing list or via the WSDL4J Tracker at SourceForge.net that can only be supported through changes to the JWSDL API, hence the need for a Maintenance Release of JSR110.

## *Proposed Changes*

All items in the proposed change log have been accepted, with feedback and modification to two items:

- item 3) Specify a WSDLFactory class via /META-INF/services
- item 6) Change semantics of PortType.getOperation method

All change log items have been moved to the "Accepted Changes" section.

## *Accepted Changes*

### 1) New method WSDLReader.readWSDL(WSDLLocator, Element)

Add a new method to the `WSDLReader` interface:

```
public Definition readWSDL(WSDLLocator locator,
                           Element definitionsElement)
                           throws WSDLException;
```

Currently when the JWSDL client application has a DOM `Document` or `Element` representing the WSDL, it must pass this to the `WSDLReader` with a base URI string. This new `readWSDL` method provides the ability to instead use a `WSDLLocator` for resolving relative URIs on imports.

The client application will invoke this method on `WSDLReader`, passing in a `WSDLLocator` object and an `org.w3c.dom.Element` object representing the `<wsdl:definitions>` element. This proposal does not add a `readWSDL(WSDLLocator, Document)` method to `WSDLReader`. If the client has an `org.w3c.dom.Document` representing the WSDL, it should use the `Document.getDocumentElement()` method obtain the `Element` to pass to this `readWSDL` method.

Raised originally on the JSR110 mailing list - http://groups.yahoo.com/group/jsr110-eg-disc/message/657.

Now captured by SourceForge WSDL4J Tracker item 1523867 - http://sourceforge.net/tracker/index.php?func=detail&aid=1523867&group_id=128811&atid=712792.

## 2) New method WSDLLocator.close()

Add a new method to the WSDLLocator interface:

```
public void close();
```

The client application can call `close()` to release any system resources used by the `WSDLLocator` object. For example, to close any open input streams. The `WSDLLocator` implementation must define the behaviour of the `close()` method.

Originally raised on the JSR110 mailing list - http://groups.yahoo.com/group/jsr110-eg-disc/message/660.

Now captured by SourceForge WSDL4J Tracker item 1523891 - http://sourceforge.net/tracker/index.php?func=detail&aid=1523891&group_id=128811&atid=712792

## 3) Specify a WSDLFactory class via /META-INF/services

Add support for client applications to specify the `WSDLFactory` implementation class name via a property file `/META-INF/services/javax.wsdl.factory.WSDLFactory`.

Currently, the static `WSDLFactory.newInstance()` method will search for a `WSDLFactory` implementation class name:

1. in a JVM system property “javax.wsdl.factory.WSDLFactory”, then
2. in the wsdl.properties file in the JRE/lib directory, then
3. let the implementation determine which factory to use (e.g. WSDL4J defaults to the `WSDLFactoryImpl` class).

Specifying the `WSDLFactory` implementation class at such a ‘system-wide’ level is not suitable for JWSDL applications running in managed environments such as J2EE containers, where a more fine grained scope is preferable (e.g. application-scope).

Under this change, the search sequence for the `WSDLFactory` implementation class name in the `newInstance()` method will be:

1. /META-INF/services/ javax.wsdl.factory.WSDLFactory property file
2. JVM system property "javax.wsdl.factory.WSDLFactory"
3. JRE/lib/wsdl.properties file
4. Default class name `WSDLFactoryImpl`

Note, this change does not modify any method signatures, but it does modify the behaviour of the `newInstance()` method, which will require updates to its Javadoc comments.

Note (2), the search sequence proposed for the maintenance review was JVM system property, then wsdl.properties file, then /META-INF/services, then default class name. Based on feedback from the review, this has been changed to the sequence shown above (i.e. /META-INF/services has been moved to the top).

This requirement is captured by SourceForge WSDL4J Tracker item 1439548 - http://sourceforge.net/tracker/index.php?func=detail&aid=1439548&group_id=128811&atid=712792

## 4) New method WSDLFactory.newInstance(String implName, Classloader loader)

Add a new static method to the `WSDLFactory` class:

```
public static WSDLFactory newInstance(String factoryImplName,
                                      ClassLoader classLoader);
```

This will allow a JWSDL client application to specify not only the `WSDLFactory` implementation class, but also the classloader to be used to load it. This approach is needed by applications such as Eclipse plugins and J2EE applications.

This requirement is captured by issue #2 on SourceForge WSDL4J Tracker item 1226908: http://sourceforge.net/tracker/index.php?func=detail&aid=1226908&group_id=128811&atid=712792

## 5) New accessor methods on Definition to retrieve imported content transparently

Add accessor methods to the `Definition` interface to 'flatten' a WSDL import tree by returning collections that represent the WSDL elements declared directly in the Definition and those declared in any imported Definitions within the <wsdl:import> tree.

Specifically, add these methods to the javax.wsdl.Definition interface:

```
public Map getAllPortTypes()
public Map getAllBindings()
public Map getAllServices()
```

The existing methods `getPortTypes()`, `getBindings()` and `getServices()` will continue to return just the directly declared elements. The new `getAllXXX` methods above will return the combined set of directly declared and imported elements.

The Map returned by these new methods will be a Map of Lists. The Map key will be a `QName` and the Map value will be a List of one or more WSDL components with that `QName`. Note that in WSDL 1.1 a definition can import another WSDL from the same target namespace, so it is possible for there to be multiple porttypes, bindings or services with the same qualified name.

Captured by SourceForge Tracker item 1526754:
http://sourceforge.net/tracker/index.php?func=detail&aid=1526754&group_id=128811&atid=712792

## 6) Change semantics of PortType.getOperation method

The semantics of this method will be changed to handle overloaded operations where some of those operations may have unnamed input and/or output elements. Note, this does not involve change to any interface or method signature, but it will require a modification to the behaviour described in the Javadoc comment for this method.

`PortType` defines the method:
`getOperation(String opName, String inputName, String outputName)`

The current behaviour of this method as described by its Javadoc is to ignore the `inputName` and `outputName` arguments from the operation search criteria if they are null. However, this means the method cannot be used with overloaded operations to explicitly request an operation with unnamed input and output elements when the porttype also contains similarly named operations that have named inputs and outputs. In this case, the operations will be searched on `opName` only and a duplicate operation error will occur.

For example, in the following WSDL fragment both operations have the same name but the first has named input and output elements while the second has unnamed input and output.

```
<wsdl:portType name="Account5">
  <wsdl:operation name="GetBalance">
    <wsdl:input name="Customer" message="tns:CustIn" />
    <wsdl:output name="Balance" message="tns:BalOut" />
  </wsdl:operation>
  <wsdl:operation name="GetBalance">
    <wsdl:input message="tns:AcctnoIn" />  <!-- null inputName -->
    <wsdl:output message="tns:BalOut" />   <!-- null outputName -->
  </wsdl:operation>
</wsdl:portType>
```

The method call `porttype.getOperation("GetBalance", null, null)` will currently return a duplicate operation error because this method will ignore the null arguments and search only for operations named “GetBalance”.

The semantics of the `getOperation` method will be changed so that null values for the `inputName` or `outputName` arguments will first trigger a search for operations that match the specified `opName` argument but have unnamed input or output and if none are found, will then search only on the `opName` ignoring any input or output names.

Note, during the maintenance review period it became apparent that because the new semantics proposed above change the behaviour of null values for the `inputName` and `outputName` method arguments as described in the existing API Javadoc, this would break backward compatibility of the API. A new approach as been adopted instead, which is to retain the existing behaviour of null arguments (i.e. ignore these arguments in the search criteria) and specify a new value ":none" for `inputName` or `outputName` to explicitly indicate a search for unnamed input or output messages. The colon prefix in ":none" will eliminate possible name clashes with input or output message names as these must be of type NCName (i.e. they cannot contain the colon ":" character). This approach was posted to the mailing list and captured in the issue Tracker.

This requirement is captured by SourceForge Tracker item 1444755:
http://sourceforge.net/tracker/index.php?func=detail&aid=1444755&group_id=128811&atid=712792

## 7) removeXXX method for each addXXX method

Ensure that all `addXXX` methods have a corresponding `removeXXX` method. This will improve support in the JWSDL API for modifying WSDL programmatically. That is, it should be possible remove anything that has been previously added.

Currently, only some of the `addXXX` methods have matching `removeXXX` methods. For example, `Definition` has 'add' and 'remove' methods for `Message`, `PortType`, `Binding` and `Service`, but the `addImport` and `addNamespace` methods do not have matching 'remove' methods.

The following `removeXXX` methods will be added to the API:

```
Definition.removeImport(Import import)

Definition.removeNamespace(String prefix)

Binding.removeBindingOperation(String name,
                               String inputName,
                               String outputName)

BindingOperation.removeBindingFault(String name)

Message.removePart(String name)

Operation.removeFault(String name)

PortType.removeOperation(String name,
                         String inputName,
                         String outputName)

Service.removePort(String name)
```

```
ElementExtensible.removeExtensibilityElement(ExtensibilityElement)

SOAPHeader.removeSOAPHeaderFault(SOAPHeaderFault)

MIMEMultipartRelated.removeMIMEPart(MIMEPart)
MIMEPart.removeExtensibilityElement(ExtensibilityElement)
```

This requirement was originally raised via the mailing list:
http://groups.yahoo.com/group/jsr110-eg-disc/message/662

It is now captured by SourceForge Tracker item 1526732:
http://sourceforge.net/tracker/index.php?func=detail&aid=1526732&group_id=128811&atid=712792

## 8) Permit extension elements and attributes for all WSDL elements

Allow all elements in the WSDL namespace to be extensible by elements or attributes from outside the WSDL namespace.

Element and attribute extensibility will be refactored into a new super-interface `javax.wsdl.WSDLElement` and all the interfaces in the `javax.wsdl` package that represent elements from the WSDL namespace will extend `WSDLElement` .

For example, the interfaces in the `javax.wsdl` package will be defined like this:

```
public interface WSDLElement extends java.io.Serializable,
                                     AttributeExtensible,
                                     ElementExtensible

public interface PortType extends WSDLElement
public interface Operation extends WSDLElement
public interface Binding extends WSDLElement
...
```

The original WSDL 1.1 schema at http://schemas.xmlsoap.org/wsdl/ specifies that certain WSDL elements may have extension elements while others may have extension attributes. Under last JSR110 maintenance release, the JWSDL 1.1 API was modified to conform to these schema constraints for WSDL extensions.

The WS-I Basic Profile 1.1 specification (http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html) relaxes these WSDL extension rules, permitting every WSDL element to be extensible by elements or attributes. In particular, the WS-I BP 1.1 Errata er007 at http://www.ws-i.org/Profiles/BasicProfile-1.1-errata.html refers to a modified WSDL 1.1 schema at http://ws-i.org/profiles/basic/1.1/wsdl-2004-08-24.xsd that permits extension elements or attributes on every WSDL element. WS-I BP 1.1 compliance is required to support web services standards like JAX-WS (JSR224).

This change to the JWSDL API will relax the original WSDL 1.1 schema constraints on WSDL extension to permit the more flexible extension permitted by WS-I BP 1.1.

The following table captures the proposed changes:

| JWSDL interface in javax.wsdl package | WSDL 1.1 schema type | Extensibility defined in original WSDL 1.1 schema and enforced in JWSDL 1.1 | Extensibility defined in WSDL 1.1 schema modified by WS-I BP 1.1 and proposed for JWSDL 1.2 |
|---|---|---|---|
| Definition | tDefinitions | Elements | Elements and Attributes |
| Import | tImport | Attributes | Elements and Attributes |
| Types | tTypes | Elements | Elements and Attributes |
| Message | tMessage | Elements | Elements and Attributes |
| Part | tPart | Attributes | Elements and Attributes |
| PortType | tPortType | Attributes | Elements and Attributes |
| Operation | tOperation | Elements | Elements and Attributes |
| Input | tParam | Attributes | Elements and Attributes |
| Output | tParam | Attributes | Elements and Attributes |
| Fault | tFault | Attributes | Elements and Attributes |
| Binding | tBinding | Elements | Elements and Attributes |
| BindingOperation | tBindingOperation | Elements | Elements and Attributes |
| BindingInput | tBindingOperationMessage | Elements | Elements and Attributes |
| BindingOutput | tBindingOperationMessage | Elements | Elements and Attributes |
| BindingFault | tBindingOperationFault | Elements | Elements and Attributes |
| Service | tService | Elements | Elements and Attributes |
| Port | tPort | Elements | Elements and Attributes |

This change is captured by SourceForge Tracker item 1526749:
http://sourceforge.net/tracker/index.php?func=detail&aid=1526749&group_id=128811&atid=712792

## 9) Soap 1.2 binding extensions

Add support for WSDL binding extensions for SOAP 1.2, as defined at http://www.w3.org/Submission/wsdl11soap12/.

A new package `javax.wsdl.extensions.soap12` will contain these interfaces:

```
SOAP12Address
SOAP12Binding
SOAP12Body
SOAP12Fault
SOAP12Header
SOAP12HeaderFault
SOAP12Operation
```

These interfaces look similar to the existing SOAP 1.1 binding extensions contained in the `javax.wsdl.extensions.soap` package, but they reflect the changes brought about by SOAP 1.2.

This change is captured by SourceForge Tracker item 1526750:
http://sourceforge.net/tracker/index.php?func=detail&aid=1526750&group_id=128811&atid=712792

## 10) Change WSDLException.toString() to return short message

Eliminate the exception stack trace from the result of `WSDLException.toString()`, instead just returning the short message as described in Java 1.4 for `java.lang.Exception.toString()`. That is, just return the exception classname `"javax.wsdl.WSDLException"`, then a colon ":", then detail message returned by `WSDLException.getMessage()`. This will be achieved by removing the `toString()` method from `WSDLException` and letting it inherit `toString()` from `java.lang.Exception`.

The format of the detailed message returned by `WSDLException.getMessage()` will be improved slightly by inserting the target throwable's classname. So the sequence of information in the detail message will now be:

WSDLException at location … : faultCode=… : [this msg] : [target classname] : [target msg]

This will eliminate the verbose trace output currently produced when a client application calls `WSDLException.toString()`. The client application can still call `printStackTrace()` if verbose trace information is required.

This is captured by SourceForge Tracker item 1526751:
http://sourceforge.net/tracker/index.php?func=detail&aid=1526751&group_id=128811&atid=712792

## 11) Change minimum Java level supported by JWSDL to Java 1.4

JWSDL currently specifies support for Java 1.2 or greater. The minimum supported Java level should now be specified as Java 1.4 as this is itself relatively old and in broad use now and it offers improvements over Java 1.2 such as the `java.lang.Exception` class from Java 1.4 which supports a wrapped target exception via the `getCause()` method.

## *Deferred Changes*

None of the proposed changes were deferred after the maintenance review.