JAKARTA EE

Jakarta SOAP with Attachments

Jakarta SOAP with Attachments Team,
https://projects.eclipse.org/projects/ee4j.jaxws

2.0, August 10, 2020:

Table of Contents

Eclipse Foundation Specification License . . . 1
Disclaimers . . . 2
Scope . . . 3
1. Introduction . . . 4
1.1. Acknowledgements . . . 4
1.2. Terminology . . . 4
2. Preface . . . 6
2.1. Audience . . . 6
2.2. Abstract . . . 6
3. Typographic Conventions . . . 7
4. Package Overview . . . 8
4.1. MessageFactory & SOAPMessage Objects . . . 8
4.2. SOAPPart & AttachmentPart . . . 8
4.3. MimeHeader(s) Objects . . . 9
4.4. SOAP Element . . . 9
4.5. SOAPEnvelope & SOAPBody objects . . . 9
4.6. SOAPBodyElement & SOAPFault . . . 10
4.7. SOAPFaultElement & Detail . . . 10
4.8. SOAPHeader & SOAPHeaderElement . . . 10
4.9. SOAPConnection & SOAPConnectionFactory . . . 10
4.10. SOAPException object . . . 11
4.11. Node & Text objects . . . 11
4.12. Name . . . 11
4.13. SOAPFactory & SOAPElementFactory . . . 11
4.14. SAAJMetaFactory . . . 11
4.15. SAAJResult . . . 12
5. Package: jakarta.xml.soap . . . 13
5.1. Description . . . 13
5.2. Discovery of SAAJ implementation . . . 14
Appendix A: References . . . 15
Appendix B: Change History . . . 16
Changes in Version 2.0 . . . 16
Changes in Maintenance Release 4 . . . 16
Changes in Maintenance Release 3 . . . 16
Changes in Maintenance Release 2 . . . 17

```
Specification: Jakarta SOAP with Attachments

Version: 2.0

Status: Final Release

Release: August 10, 2020
```


Eclipse Foundation Specification License

By using and/or copying this document, or the Eclipse Foundation document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the Eclipse Foundation document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- link or URL to the original Eclipse Foundation document.
- All existing copyright notices, or if one does not exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright (c) [$date-of-document] Eclipse Foundation, Inc. [url to this license]"

Inclusion of the full text of this NOTICE must be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of Eclipse Foundation documents is granted pursuant to this license, except anyone may prepare and distribute derivative works and portions of this document in software that implements the specification, in supporting materials accompanying such software, and in documentation of such software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

The notice is:

"Copyright (c) 2018 Eclipse Foundation. This software or document includes material copied from or derived from [title and URI of the Eclipse Foundation specification document]."

Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE COPYRIGHT HOLDERS AND THE ECLIPSE FOUNDATION WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of the copyright holders or the Eclipse Foundation may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

Scope

Jakarta SOAP with Attachments defines an API enabling developers to produce and consume messages conforming to the SOAP 1.1, SOAP 1.2, and SOAP Attachments Feature.

Chapter 1. Introduction

This specification was developed following the Java™ Community Process (JCP2.10). Comments from experts, participants, and the broader developer community were reviewed and incorporated into this specification.

The SAAJ Specification, version 1.1 was a maintenance release of the Java™ API for XML Messaging (JAXM) 1.0 specification. JAXM 1.0 was the final deliverable of JSR067 Expert Group (EG). The proposed changes specified in the JSR067 changelog and accepted on 15 April 2002, have been incorporated into this document.

The proposed changes specified in the second JSR067 changelog and accepted on 23 April 2003, have been incorporated into this document as SAAJ Specification, version 1.2.

The proposed changes specified in the third JSR067 changelog have been incorporated into this document as SAAJ Specification, version 1.3.

The proposed changes specified in the fourth JSR067 changelog have been incorporated into this document as SAAJ Specification, version 1.4.

Starting with Jakarta SOAP Attachments Specification, Version 2.0, this document is developed under Jakarta EE Specification Process

1.1. Acknowledgements

This maintenance release is the product of collaborative work within the Java community.

1.2. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 as quoted here:

MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

MUST NOT: This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the

full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

MAY: This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.

Chapter 2. Preface

2.1. Audience

This document is intended for developers using the Java™ programming language who wish to produce and consume messages conforming to the SOAP 1.1, and SOAP 1.2 specification and the SOAP Attachments Feature.

Familiarity with the SOAP specifications (including the associated processing model), MIME standards, and XML is assumed.

2.2. Abstract

Jakarta SOAP with Attachments defines an API enabling developers to produce and consume messages conforming to the SOAP 1.1, SOAP 1.2 and SOAP Attachments Feature.

In the interest of backward compatibility, SAAJ continues to offer a client-side communication capability enabling developers to communicate in a point-to-point and request-response manner with SOAP services bound to HTTP. This communication capability, within the context of the SAAJ specification, is optional. However, specifications depending on SAAJ are free to require support for the SOAP to HTTP binding.

Chapter 3. Typographic Conventions

Typeface[1]	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
AaBbCc123	What you type, when contrasted with on-screen computer output	% **su** `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide* . These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename* .

[1] The settings on your browser might differ from these settings.

Chapter 4. Package Overview

This chapter presents an overview of the SAAJ which consists of the single package; `jakarta.xml.soap` . The intent here is to provide an overview of the package only, the details of which can be found in the following chapter.

The `jakarta.xml.soap` package provides the primary abstraction for SOAP Messages with MIME attachments. Attachments may be entire XML documents, XML fragments, images, text documents, or any other content with a valid MIME type. In addition, this package provides a simple client-side view of a request-response style of interaction with a SOAP service.

4.1. MessageFactory & SOAPMessage Objects

The `MessageFactory` class is used to create `SOAPMessage` objects. Clients may create SOAPMessage objects by calling the `MessageFactory.createMessage` method.

The `SOAPMessage` class is the root class for all SOAP messages. Such messages must contain a single `SOAPPart` object and may contain one or more `AttachmentPart` objects. The "on-the-wire" encoding of a SOAP message is governed by whether the `SOAPMessage` object includes `AttachmentPart` objects. If it does, the `SOAPMessage` object is encoded as a MIME message otherwise it is encoded as a simple XML message. Attachments may contain data of any type including XML. The SOAPPart is always XML.

SAAJ allows for creation and consumption of both SOAP 1.1 and SOAP 1.2 messages by introducing the notion of Protocol aware MessageFactories. The protocol here refers to a particular version of SOAP. For example a SOAP 1.2 aware MessageFactory can be obtained by calling the MessageFactory.newInstance method and passing it the appropriate protocol identifier. The allowed protocol identifiers have been defined in SOAPConstants. For processing incoming messages a special protocol identifier called DYNAMIC_SOAP_PROTOCOL can be used to allow a Node to accept both SOAP 1.1 and SOAP 1.2 messages.

4.2. SOAPPart & AttachmentPart

The `SOAPPart` object is a MIME part containing the `SOAPEnvelope` object. The `SOAPEnvelope` object must contain a single `SOAPBody` object and may contain a `SOAPHeader` object.

A `SOAPMessage` object may contain zero or more `AttachmentPart` objects. Each `AttachmentPart` object in turn contains application-specific content and corresponding MIME headers. The MIME headers consist of name/value pairs that are used to identify and describe the content. For MIME content-types of `text/plain` , `text/html` and `text/xml` , the `DataContentHandler` object performs the necessary conversions to and from the Java types corresponding to the MIME types. Other MIME types can be supported by passing an `InputStream` object (that contains the content data) to the `AttachmentPart.setContent` method. Similarly, the contents and header from an `AttachmentPart` object can be retrieved using the `getContent` method. Depending on the `AttachmentPart` objects present, the returned `Object` can be either a typed Java object corresponding to the MIME type or an `InputStream`

object that contains the content as bytes. The `clearContent` method is a helper method intended to facilitate the removal of all the content from an `AttachmentPart` object while leaving the header information.

A SAAJ implementation must support the following MIME types. Additional MIME types may be supported using the `jakarta.activation.DataHandler` class and the Jakarta Activation Framework.

Table 1. SAAJ supported MIME types

MIME Type	Java Type
`text/plain`	`java.lang.String`
`multipart/*`	`jakarta.mail.internet.MimeMultipart`
`text/xml` or `application/xml`	`javax.xml.transform.Source`

SAAJ provides methods for setting and getting the Raw content of an Attachment. Methods have also been provided to get the content as Base64 encoded character data. Additionally a getAttachment method on the SOAPMessage provides for retrieval of an Attachment referenced from a SOAPElement using an href attribute as described in SOAP Messages with Attachments, or via a single Text child node containing a URI as described in the WS-I Attachments Profile 1.0 for elements of schema type ref:swaRef

4.3. MimeHeader(s) Objects

The `MimeHeaders` class is a container for `MimeHeader` objects and serves as an abstraction for the MIME headers that must be present if an `AttachmentPart` object exists in a `SOAPMessage` object.

The `MimeHeader` object is the abstraction for a name/value pair of a MIME header. A `MimeHeaders` object may contain one or more `MimeHeader` objects.

4.4. SOAP Element

The `SOAPElement` object is the base class for all of the classes that model the SOAP objects defined by the SOAP1.1 and SOAP 1.2 specifications. A `SOAPElement` object may be used to model the following:

- content in a `SOAPBody` object
- content in a `SOAPHeader` object
- content that can follow the `SOAPBody` object within a `SOAPEnvelope` object
- whatever may follow the detail element in a `SOAPFault` object

4.5. SOAPEnvelope & SOAPBody objects

The `SOAPEnvelope` object is a container object for the `SOAPHeader` and `SOAPBody` portions of a `SOAPPart` object. The `SOAPEnvelope` object must contain a `SOAPBody` object, but the `SOAPHeader` object is optional.

The SOAPEnvelope and SOAPBody objects both extend the SOAPElement object. The SOAPBody object models the contents of the SOAP body element in a SOAP message. A SOAP body element contains XML data that may determine how application-specific content must be processed.

4.6. SOAPBodyElement & SOAPFault

SOAPBody objects contain SOAPBodyElement objects that model the content of the SOAP body. An example of a SOAPBodyElement is the SOAPFault object.

4.7. SOAPFaultElement & Detail

The SOAPFaultElement is used to represent the contents of a SOAPFault object.

The Detail interface is a container for DetailEntry objects that provide application-specific error information associated with the SOAPBody object that contains it.

A Detail object is part of a SOAPFault object and may be retrieved using the getDetail method of the SOAPFault object.

The DetailEntry object extends SOAPElement and models the contents of a Detail object.

4.8. SOAPHeader & SOAPHeaderElement

A SOAPHeader object is an abstraction of the SOAP header element. A SOAPHeader object can be created using the SOAPEnvelope.addHeader method. SOAPHeader objects can have only SOAPHeaderElement objects as their immediate children. The addHeaderElement method creates a new HeaderElement object and adds it to the SOAPHeader object.

SOAPHeader and SOAPHeaderElement objects both extend the SOAPElement object. A SOAPHeaderElement object models the contents of the SOAP header of a SOAP envelope.

4.9. SOAPConnection & SOAPConnectionFactory

The SOAPConnection object represents a simple client-side view of a request-response style of SOAP messaging. A SAAJ client may choose to establish a synchronous point-to-point connection to a SOAP service using the createConnection method of the SOAPConnectionFactory object. Subsequently, a SOAPMessage may be sent to a remote party using the call method on the SOAPConnection object. Note that the call method will block until a SOAPMessage object is received.

A SAAJ based application may choose to use the call method to implement the client side of a simple point-to-point synchronous one-way message exchange scenario. In such a case, it is the application's responsibility to ignore the SOAPMessage object returned by the call method because the SOAPMessage object's only purpose is to unblock the client. It is assumed that a one-way service will not return a response to a request using the same connection when the SOAPConnection.call method was used to

send the request.

SAAJ also provides support for the SOAP 1.2 Response Message Exchange Pattern (http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapresmep) via the `SOAPConnection.get` method. This method can be used for pure information retrieval, where the representation of an available resource, identified by a URI, is fetched using a HTTP GET request without affecting the resource in any way.

4.10. SOAPException object

The `SOAPException` object extends `java.lang.Exception` and is used to signal SOAP level exceptions.

4.11. Node & Text objects

The `Node` object models a node (element) of a DOM abstraction of an XML document.

The `Text` object extends `Node` and represents a node whose value is text. A `Text` object may model either text that is content or text that is a comment.

4.12. Name

The `Name` object models an XML name. This interface provides methods for getting the local names, namespace-qualified names, the prefix associated with the namespace for the name, and the URI of the namespace.

Name objects are created using the `SOAPEnvelope.createName` method.

4.13. SOAPFactory & SOAPElementFactory

These factories are intended primarily for the use of application components or tools that require the capability of inserting XML fragments into a SOAP Message. In SAAJ v1.1, the `SOAPElementFactory` has been deprecated in favor of `SOAPFactory` which serves as a super factory for the creation of `SOAPElement` , `Name` , and `Detail` objects.

4.14. SAAJMetaFactory

This Factory is the access point for the implementation classes of all the other factories defined in the SAAJ API. All of the newInstance methods defined on factories in SAAJ defer to instances of this class to do the actual object creation. The implementations of newInstance() methods (in SOAPFactory and MessageFactory) that existed in SAAJ 1.2 have been updated to also delegate to the SAAJMetaFactory when the SAAJ 1.2 defined lookup fails to locate the Factory implementation class name.

`SAAJMetaFactory` is a service provider interface. There are no public methods on this class.

4.15. SAAJResult

This concrete class acts as a holder for the results of a JAXP transformation or a Jakarta XML Binding marshalling, in the form of a SAAJ tree. This class will make it easier for the end user when dealing with transformations in situations where the result is expected to be a valid SAAJ tree. The results can be accessed by using the getResult method.

Chapter 5. Package: jakarta.xml.soap

5.1. Description

Provides the API for creating and building SOAP messages. This package is defined in the Jakarta SOAP with Attachments (SAAJ) specification.

The API in the `jakarta.xml.soap` package allows you to do the following:

- create a point-to-point connection to a specified endpoint
- create a SOAP message
- create an XML fragment
- add content to the header of a SOAP message
- add content to the body of a SOAP message
- create attachment parts and add content to them
- access/add/modify parts of a SOAP message
- create/add/modify SOAP fault information
- extract content from a SOAP message
- send a SOAP request-response message

In addition the APIs in the `jakarta.xml.soap` package extend their counterparts in the `org.w3c.dom` package. This means that the `SOAPPart` of a `SOAPMessage` is also a DOM Level 2 `Document`, and can be manipulated as such by applications, tools and libraries that use DOM (see http://www.w3.org/DOM/ for more information). It is important to note that, while it is possible to use DOM APIs to add ordinary DOM nodes to a SAAJ tree, the SAAJ APIs are still required to return SAAJ types when examining or manipulating the tree. In order to accomplish this the SAAJ APIs (specifically `SOAPElement.getChildElements()`) are allowed to silently replace objects that are incorrectly typed relative to SAAJ requirements with equivalent objects of the required type. These replacements must never cause the logical structure of the tree to change, so from the perspective of the DOM APIs the tree will remain unchanged. However, the physical composition of the tree will have changed so that references to the nodes that were replaced will refer to nodes that are no longer a part of the tree. The SAAJ APIs are not allowed to make these replacements if they are not required so the replacement objects will never subsequently be silently replaced by future calls to the SAAJ API.

What this means in practical terms is that an application that starts to use SAAJ APIs on a tree after manipulating it using DOM APIs must assume that the tree has been translated into an all SAAJ tree and that any references to objects within the tree that were obtained using DOM APIs are no longer valid. Switching from SAAJ APIs to DOM APIs is not allowed to cause invalid references and neither is using SAAJ APIs exclusively. It is only switching from using DOM APIs on a particular SAAJ tree to using SAAJ APIs that causes the risk of invalid references.

5.2. Discovery of SAAJ implementation

There are several factories defined in the SAAJ API to discover and load specific implementation:

- SOAPFactory
- MessageFactory
- SOAPConnectionFactory
- SAAJMetaFactory

First three define newInstance() method which uses a common lookup procedure to determine the implementation class:

- Checks if a system property with the same name as the factory class is set (e.g. `jakarta.xml.soap.SOAPFactory`). If such property exists then its value is assumed to be the fully qualified name of the implementation class. This phase of the look up enables per-JVM override of the SAAJ implementation.
- Use the configuration file "jaxm.properties". The file is in standard `Properties` format and typically located in the `conf` directory of the Java installation. It contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the default loading mechanism.
- Finally, if all the steps above fail, `SAAJMetaFactory` instance is used to locate specific implementation (for `MessageFactory` and `SOAPFactory`) or platform default implementation is used (`SOAPConnectionFactory`). Whenever `SAAJMetaFactory` is used, its lookup procedure to get actual instance is performed.

Appendix A: References

For more information, refer to the following web sites:

1. SOAP 1.1 http://www.w3.org/TR/SOAP
2. SOAP 1.2 http://www.w3.org/TR/soap12-part1/
3. SOAP Messages with Attachments http://www.w3.org/TR/SOAP-attachments, http://www.w3.org/TR/soap12-af
4. Jakarta Activation 2.0 https://jakarta.ee/specifications/activation/2.0/
5. Java API for XML Processing Version 1.6 https://jcp.org/en/jsr/detail?id=206#orig
6. Java API for XML Messaging Version 1.1 Final Release https://jcp.org/en/jsr/detail?id=67
7. WS-I Attachments Profile 1.0 http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html

Appendix B: Change History

Changes in Version 2.0

- Changed specification version and license.
- Changed package name to `jakarta.xml.soap`

Changes in Maintenance Release 4

The fourth maintenance release of SAAJ, SAAJ 1.4, clarifies and extends the SAAJ 1.3 specification.The goal of this maintenance release proposal is primarily to adopt changes in Java SE Platform and to make a few corrections and clarifications on the existing SAAJ 1.3 APIs. The proposed API changes in SAAJ 1.4 are backward compatible with SAAJ 1.3 APIs.

The "accepted changes", as specified in JSR067 changelog, have been incorporated in this document. The key changes are as follows:

- Changed the definition of the discovery process of SAAJ implementation.
- Added support for generics where applicable.
- Changed MAY to MUST in jakarta.xml.soap.SOAPElements methods.
- Several corrections and clarifications have been made to the JavaDocs for the API.

Changes in Maintenance Release 3

The third maintenance release of SAAJ, SAAJ 1.3, clarifies and extends the SAAJ 1.2 specification.The goal of this maintenance release proposal is primarily to provide support for SOAP version 1.2 Message Constructs and to make a few corrections and clarifications on the existing SAAJ 1.2 APIs. The proposed API changes in SAAJ 1.3 are backward compatible with SAAJ 1.2 APIs.

SOAP version 1.2 has a number of changes in syntax and provides additional (or clarified) semantics from those described in SOAP 1.1. This proposed changes in this maintenance release are concerned with the following areas:

- Support for SOAP version 1.2 message constructs in the API.
- Factoring out the creation of all SAAJ Factory classes into a single SPI that allows creation of SOAP version aware Factories.
- Addition of a few new classes and new methods in certain existing classes and interfaces.
- Support for overloaded QName based methods in certain classes and interfaces.
- Clarification of semantics and correction of wording of JavaDocs and specification

A brief summary of the proposed changes follows:

- Support for SOAP Version 1.2 message constructs in the API: SOAP Version 1.2 has a number of changes in syntax and introduces several new Message Constructs. SAAJ 1.3 will support SOAP Version 1.2 Message Constructs.
- SPI for Creation of Factory Instances: SAAJ 1.3 will support SOAP Version 1.2 Message Constructs, while at the same time being backward compatible in its support for SOAP Version 1.1. We would like to define an SPI (SAAJMetaFactory) for factoring out the creation of SOAP Version aware Factory classes into a single place. Changing out the SAAJMetaFactory has the effect of changing out the entire SAAJ implementation. Backward compatibility is maintained by ensuring that the default protocol is set to SOAP Version 1.1.
- Definition of new Class SAAJResult: A SAAJResult object acts as a holder for the results of a JAXP transformation or a JAXB marshalling, in the form of a SAAJ tree. This class will make it easier for the end user when dealing with transformations in situations where the result is expected to be a valid SAAJ tree.
- Addition of overloaded methods which accept a QName instead of a Name: QName is the preferred representation of XML qualified names, and hence we would like to introduce overloaded methods in all APIs where a corresponding method was accepting a jakarta.xml.soap.Name as argument. The Name interface may be deprecated in a future release of SAAJ in favor of QName.
- Clarify and correct the wording of JavaDocs and specification: None of these changes will break backward compatibility for SOAP 1.1 users. Corrections of this nature cost little and improve the overall integrity of the specification making correct implementations easier to create, validate and use.
- Addition of new methods in certain Interfaces and Classes: A few new methods have been introduced in AttachmentPart, SOAPBody, and SOAPElement. These new methods are intended for ease of use and to assist SAAJ users when dealing with some of the newer SOAP features.
- Making SOAPPart a jakarta.xml.soap.Node: The SOAPPart is also a SOAP Node.

Deferred Changes:

- The deprecation of Name Interface has been deferred to a later release.
- DOM Level 3 Support: Implementations of SAAJ 1.3 must provide support for DOM Level 3 APIs.

Changes in Maintenance Release 2

The second maintenance release of SAAJ, SAAJ 1.2, clarifies and extends the SAAJ 1.1 specification. The "accepted changes", as specified in the Change Log forSOAP with Attachments API for Java™, have been incorporated into this document. A summary of the changes follows:

- The core SAAJ classes and interfaces: `Node` , `SOAPElement` , `SOAPPart` , and `Text` now extend the equivalent interfaces in the `org.w3c.dom` package: `Node` , `Element` , Document and `Text` respectively.
- The ability to get and set properties on `SOAPMessage` has been added to `SOAPMessage` in order to facilitate extensibility and two new properties have been added in order to take advantage of this extensibility: `CHARACTER_SET_ENCODING` allows the character encoding to be set to "utf-8" or "utf-16"

where "utf-8" is the default. Implementations may optionally support other character encodings. `WRITE_XML_DECLARATION` allows clients to specify whether or not an XML Declaration will be written at the start of the SOAP part of the message. The valid values are "true" and "false" with "false" being the default.

- Several APIs have been extended in order to provide greater ease of use. The `Node` interface has gained a `setValue` () method. `SOAPFault` has been enhanced with several methods that facilitate the handling of its sub-elements. `SOAPMessage` , `SOAPElement` , `SOAPBody` and `SOAPHeader` have all been given new methods that enhance navigation of the tree. A removeContents() element has been added to `SOAPElement` in order to assist in the construction of messages that contain a fault.
- Several corrections and clarifications have been made to the JavaDocs for the API.

This specification has been derived from the `jakarta.XML.SOAP` package originally defined in the JAXM 1.0 specification. The "accepted changes," as specified in JSR067 changelog, have been incorporated in this document. The key changes are as follows:

- `jakarta.xml.soap` package was moved from the JAXM specification to this document. In the interest of consistency and for simplifying synchronization of specifications, this document has been designated as version 1.1 of the SAAJ specification. There are no prior versions of the SAAJ specification.
- The `call` method signature of the `SOAPConnection` object has been modified so as to remove the dependency of SAAJ on JAXM.
- The `newInstance` method of `SOAPConnectionFactory` may throw an `UnsupportedOperationException` hence making the implementation of the `SOAPConnection.call()` functionality optional.
- The `SOAPElementFactory` has been deprecated and a new "super" factory for creating `Element` , `Detail` , and `Name` objects created. The previous `SOAPElementFactory` methods now delegate to the appropriate `SOAPFactory` methods.