Java™ API for XML Messaging (JAXM) v0.92

Please send technical comments to: jaxm-expertgroup@eng.sun.com
Please send business comments to: jaxm-business@eng.sun.com

Public Review DRAFT– June 2001

Nicholas Kassem <Nick.Kassem@eng.sun.com>
Anil Vijendran <Anil.Vijendran@eng.sun.com>
Rajiv.Mordani <Rajiv.Mordani@eng.sun.com>

Java™ API for XML Messaging Specification ("Specification")
Version: 0.92
Status: Public Review Draft 1
Release: June, 2001

RESTRICTED RIGHTS LEGEND.

If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT.
You may wish to report any ambiguities, inconsistencies, or inaccuracies you may find in connection with your evaluation of the Specification ("Feedback"). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

Contents

Status

JAXM.S.1 Status of this document

This specification is being developed following the Java™ Community Process (JCP2.0). Comments from Experts, Participants, and the broader Java™ Developer community will be reviewed and incorporated into this specification.

This document is the version of the JAXM 0.92 Specification. The main goal of this draft is to enable the Developer community to review and comment on the work of the JSR067 Expert Group (EG).

This document has been designated as Public Review Draft 1.

JAXM.S.2 Acknowledgements

Although this specification is still work-in-progress it is appropriate to acknowledge the efforts and collective wisdom of the JSR067 Expert Group and the companies who are supporting their participation in the Java™ Community.

"Simeon Simeonov" <simeons@allaire.com>
"Dan Frantz" <dan.frantz@bea.com>
"Pal Takacsi-Nagy" <pal.takacsi@bea.com>
"Bill Jones" <BJones@Bluestone.com>
"Krishna Sankar" <ksankar@cisco.com>
"Jean-Jacques Dubray" <jjdubray@exceloncorp.com>
"Daniel Haller" <daniel_haller@hp.com>
"Scott Hinkelman" <srh@us.ibm.com>
"David Clay" <david.clay@oracle.com>
"Neal Wyse" <neal.wyse@oracle.com>

"David A. Chappell" <chappell@progress.com>
"Colleen Evans" <cevans@progress.com>
"Larry Cable" <larry.cable@sun.com>
"Jean Choi" <Jean.Choi@sybase.com>
"Markus Breilmann" <Markus.Breilmann@tamgroup.com>
"Waqar Sadiq" <wsadiq@vitria.com>
"Francis Ho" <francis.ho@webMethods.com>
“Peter Eberlein” <peter.eberlein@sapmarkets.com>
"Prasad Yendluri" <pyendluri@webmethods.com>

JAXM.S.3 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”, means that the definition is an absolute requirement of the specification.

MUST NOT: This phrase, or the phrase “SHALL NOT”, means that the definition is an absolute prohibition of the specification.

SHOULD: This word, or the adjective “RECOMMENDED”, means 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”, means 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.)

Preface

JAXM.P.1 Audience

This document is intended for Java™ Developers, e-commerce Architects, and application developers focused on the developement of Business-to-Business messaging applications. Many of the concepts mentioned in this document are being discussed by participants within the ebXML (http://www.ebxml.org) community and the W3C XML Protocol working group.

JAXM.P.2 Abstract

The Java™ API for XML Messaging (JAXM v0.92) enables the Java™ Community to develop business applications that support emerging industry messaging standards based on the SOAP1.1(with attachments) specifications. Since the XML messaging standards are being developed outside of the Java™ Community Process and are evolving at different rates (driven by a diverse set of business and technical requirements), the JAXM 0.92 specification does not mandate the usage of any specific XML messaging standard. The term standard is being used here to denote a specific usage of SOAP messaging. Within the context of this document the term "Profile" is used to refer to a specific prtocol based on SOAP1.1 (w/Attachments).

A JAXM compatible implementation must therefore, support SOAP1.1 [See “SOAP” on page 111], SOAP With Attachments [See “SOAP Messages with Attachments” on page 111], and may support one or more SOAP Message Profiles [See “Messaging Profiles” on page 12]. This specification makes no

assumption about the number of such Profiles but assumes that they must be named in a consistent and standard way.

CHAPTER JAXM.1

Background

JAXM.1.1 Conceptual model

The following figure presents a conceptual relationship between JAXM and other architectural elements required in web-based, Business-to-Business Messaging.

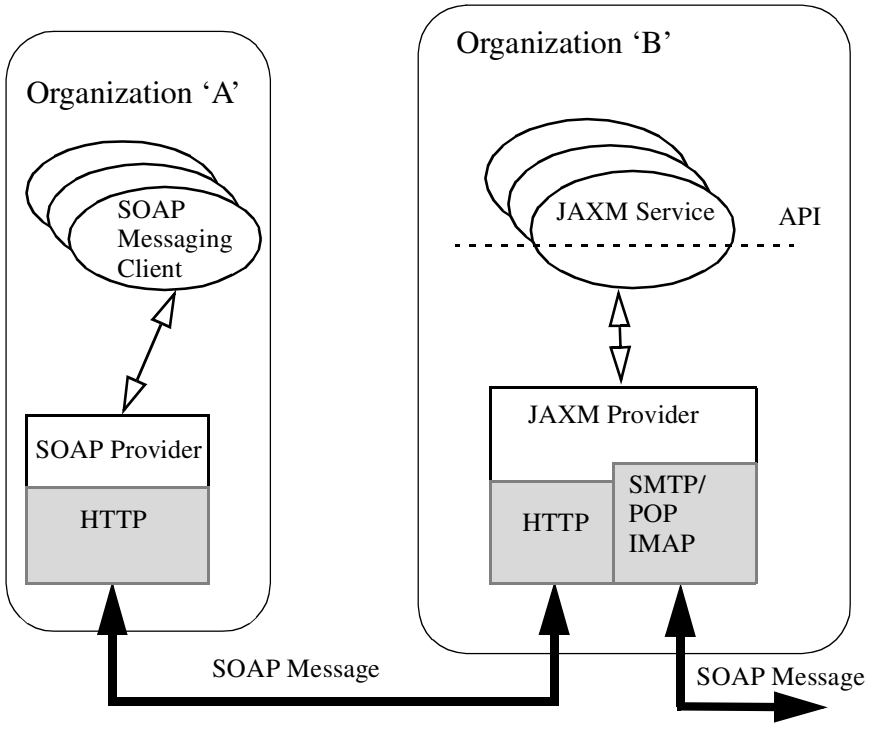

Figure 1. B2B Messaging Conceptual Model

JAXM is intended to be a lightweight messaging API for the development of XML based business Messaging applications. These applications appear primarily, though not exclusively, at the edge of organizations. The term “edge” is being used loosely to denote the set of applications that, collectively, deal with the production and consumption of **standard** business messages. The requirement for processing such messages is being fueled by the increasing need of organizations, irrespective of their size, to electronically exchange business documents. An application designed to consume specific business documents in an agreed-upon manner, and in response, produce appropriate business documents, is informally referred to as a business service. Such services, when deployed in a Web Container are typically called Web Services. The formal specification of such services is outside the scope of this document.

The previous figure makes a logical distinction between the implementation of the JAXM API, and a JAXM Provider. The latter is responsible for the actual transmission and reception of SOAP messages. An application that is written to the JAXM API is referred to as a JAXM Client.

JAXM.1.2 Scope

JAXM message exchange scenarios may be synchronous or asynchronous in nature but are always document centric. The exchange of XML documents between business partners are assumed to occur in a choreographed manner. The use-cases associated with JAXM fall into five major categories and all JAXM implementations must support these interaction styles. Note that, for simplicity, the term Sender is being used to collectively refer to a JAXM Client/Provider pairing in a message production role. Similarly, the term Receiver refers to a Client/Provider coupling in a messsage consumption role.

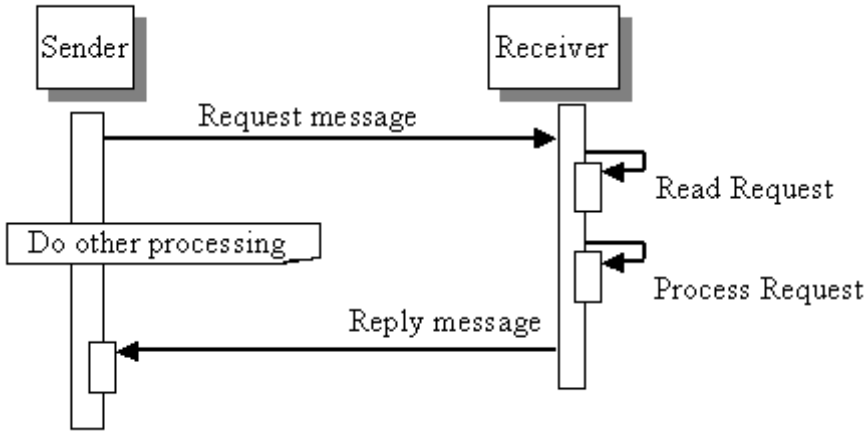

Figure 2. Asynchronous Inquiry

In this scenario, the Sender is assumed to send a message without needing to wait for a response. The Receiver is required to Read and Process the request and generate an appropriate Reply to the original Request. The time interval between a Request and a Reply may be measured in days. JAXM implementations must therefore be able to support long-lived transactions.

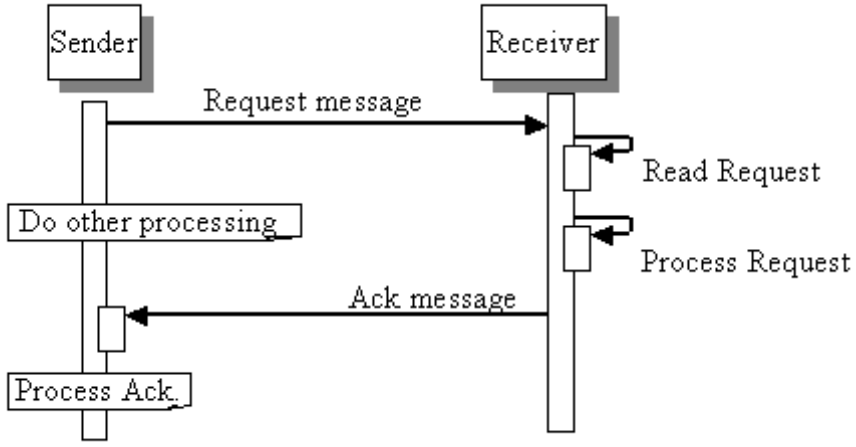

Figure 3. Asynchronous update with Acknowledgement

The previous figure depicts a scenario in which the reception of an Acknowledgement message denotes the successful completion of an earlier request. Note that JAXM does not specify how an application must correlate a Received message to a Request message sent previously.

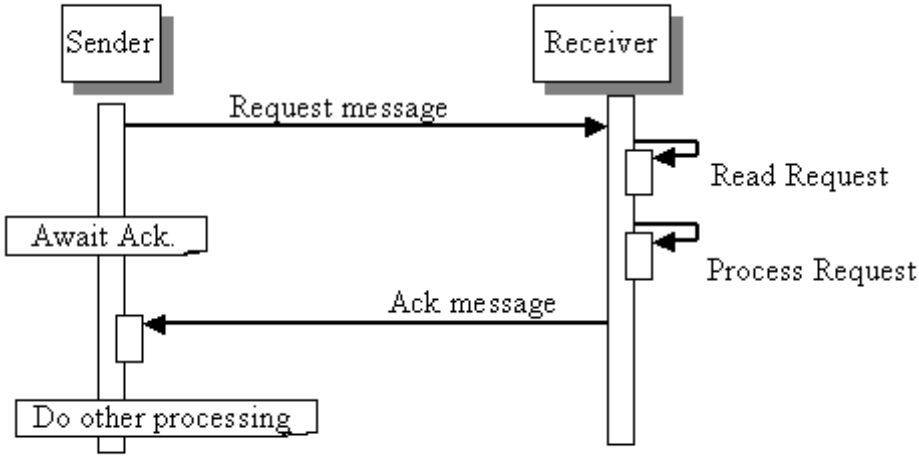

Figure 4. Synchronous Update

The figure above reflects a scenario in which the Sender either can not or must not proceed until a response to a previous message is received. The reception of an Ack message typically implies the successful completion of an earlier request.

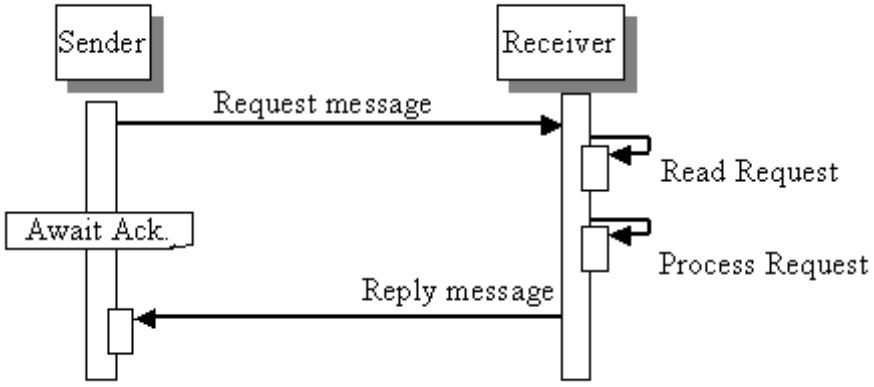

Figure 5. Synchronous Inquiry

This scenario is a simple variation on the previous case. The Sender waits for a Reply message to a previous Request. The distinction is this case is that the Reply message is typically purely informational in nature.

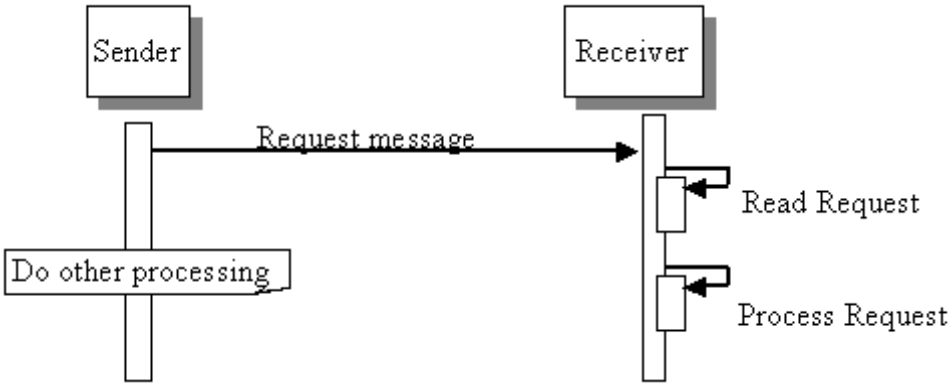

Figure 6. Fire and Forget

This last case implies that the Sender is not expecting a business level response to an earlier Request.

JAXM may facilitate the automation of only portions of an overall Business Process. The applicability of JAXM to the larger business system, is therefore a function of an overall Business Process model specific to a particular group of trading partners or vertical industries. The ways in which business objects are expressed in XML are not addressed by this specification.

JAXM.1.3 Interoperability

An important notion presented in the “B2B Messaging Conceptual Model” on page 1, is that a JAXM enabled Client must be capable of interoperating with a peer business application irrespective of whether the application is implemented using JAXM. One of the key ingredients enabling standards-based interoperability is the wide-spread adoption of a transport neutral packaging model and agreements on message header structures, manifests etc. Although JAXM is heavily biased towards using industry standards, JAXM 0.92 providers are

required to only support SOAP1.1 and SOAP with Attachments. In addition, JAXM providers may choose to support higher level industry messaging protocols built on top of SOAP messaging. A specific industry usage of SOAP is referred to here as a Profile. A JAXM Profile therefore represents a given industry or standards group's usage of SOAP.

JAXM providers must support the HTTP protocol but may also choose to implement other standard networking protocols e.g. FTP and SMTP(IMAP, POP). In all cases however, JAXM assumes SOAP messages are being transported. JAXM providers must therefore implement their Transport bindings in accordance with the SOAP 1.1 specifications.

JAXM providers may choose to support multiple, concurrent networking transports. JAXM clients, i.e. business applications, should be kept isolated from the specifics, and idiosyncrasies of underlying network transports.

JAXM.1.4 SOAP Packaging model

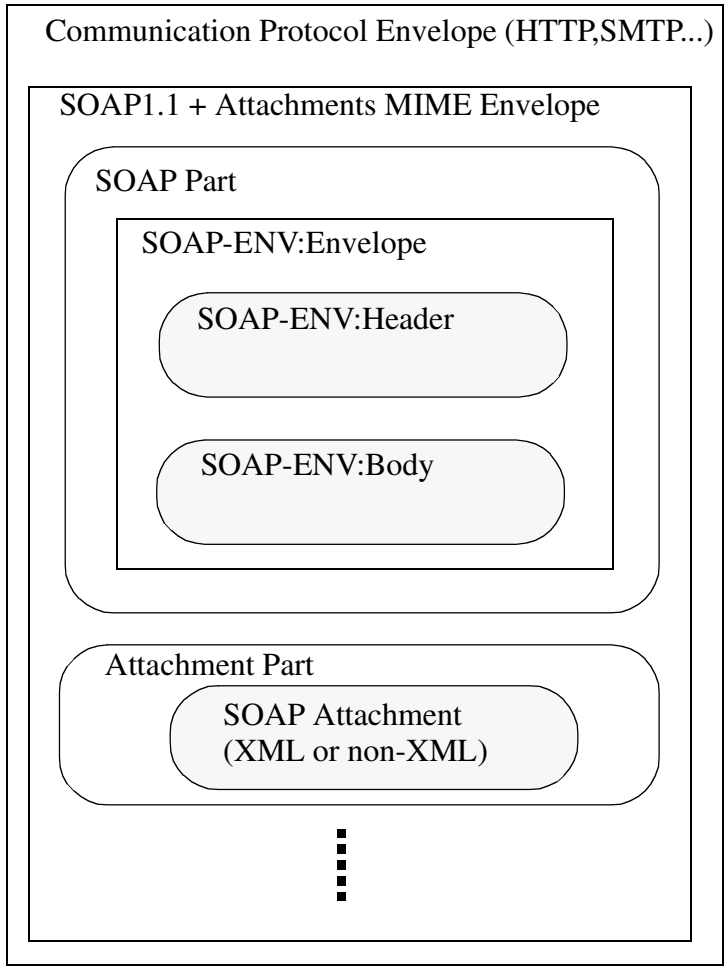

Figure 7. SOAP1.1 with Attachments

The previous figure depicts the conceptual model of an overall JAXM message. This message is aligned with the SOAP1.1 (w/attachment) specification. JAXM provides a standard way of both producing and consuming SOAP messages with or without attachments.

JAXM implementations must support SOAP Attachments. However, a JAXM client may optionally choose to create and/or consume SOAP attachments based on application specific requirements. It is the responsibility of a JAXM client to recognize the presence of one or more attachments and to process them in an application specific way. JAXM uses JAF "Java Activation Framework Version 1.0a" [see page 111] to support a flexible way of handling SOAP Attachments based on the MIME Types.

JAXM implementations must used MIME based packaging only in cases where a JAXM client chooses to send application specific data as attachments. The choice of packaging models is therefore implicitly under the control of the application developer.

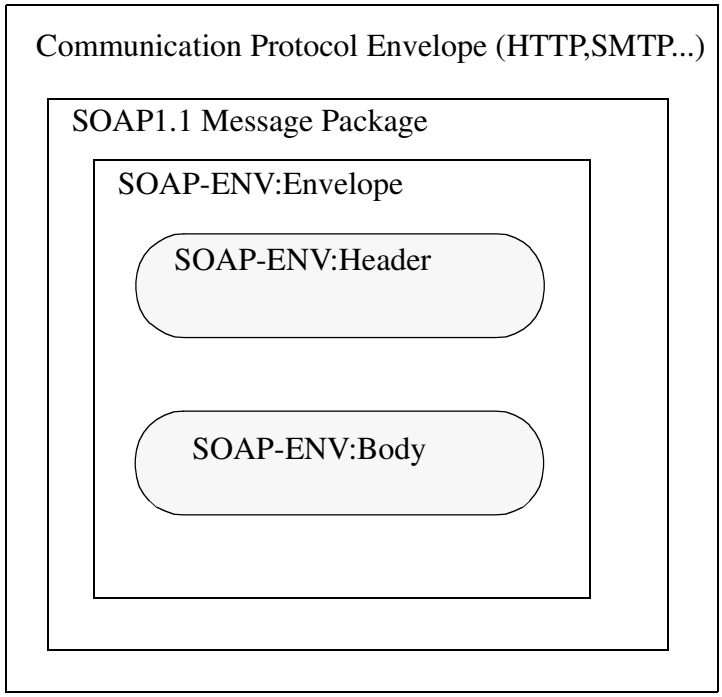

Figure 8. SOAP1.1 Packaging model

The earlier diagram "SOAP1.1 with Attachments" on page 7 assumes the presence of one or more SOAP attachments. Where an application specific SOAP attachment is not present (i.e. no attachments were specified by the developer), the

MIME Multipart/Related outer wrapper is redundant and a JAXM implementation must not produce one. The previous Figure shows the standard SOAP1.1 packaging model without attachments.

CHAPTER JAXM.2

Infrastructure

The term infrastructure is being used to refer to the implementation specific functionality required to support a JAXM implementation.

JAXM.2.1 JAXM Client

The term “client” is used here in a manner which is essentially synonemous with the term application i.e. a collection of application level functionality which is typically deployed either in a J2EE Web Container or a J2EE EJB Container. A JAXM Client is a consumer of services offered by a Provider. The capabilities of Providers may vary widely but their specifics e.g. in the area of Quality Of Service fall outside the scope of this document. JAXM Clients must be capabale of consuming SOAP1.1 (w/attachments) messages. In addition, when a JAXM Client is deployed in a J2EE Web Container the SOAP1.1 protocol must be bound to HTTP. However, the way in which JAXM Clients communicate with JAXM Providers is considered to be a private implementation detail. A JAXM Client is neither required to be co-located with its’ Provider nor is it required to be in a different Virtual Machine.

The relationship between JAXM Clients is fundamentally a peer-to-peer one, i.e. from a programming model perspective a JAXM Client may choose to play a Client role or a Server role. In addition, depending on a specific context or messaging choreography, a JAXM Client is free to switch roles. By way of example, a JAXM application may send one or more Purchase Orders to a specific Purchase Order consolidation service and may choose to wait for Receipt Acknowledgments from that service. In the event that acknowledgments fail to

arrive, the Client may choose to either resend the POs or alternatively explicitly request Acknowledgments.

JAXM.2.2 Error Messages

JAXM implementations must adopt a best effort strategy for ensuring the validity of messages produced and sent to peer entities. JAXM providers, on receiving a malformed message, are responsible for producing an appropriate error message to the offending peer-entity. The structure and addressing of inter-provider error messages is Profile specific. JAXM does not make a distinction between Error messages and any other profile-specific message. Given that Providers are "Profile-aware", they may choose to map an error condition onto a Profile specific Error message. Such messages, would be delivered to a Clients' Endpoint in the same manner as any other message. It is the responsibility of the JAXM Client to consume these messages and take application specific corrective action.

JAXM.2.3 Messaging Profiles

JAXM implementations must support SOAP1.1 and SOAP Messages with Attachments. However, these specifications provide a very basic packaging model and offer no specific addressing scheme or message structure for the routing of messages between peer entities.

A Profile represents a specific usage of a SOAP Header. JAXM does not specify what specific XML content must be placed in the SOAP Header, Body or Attachments. Most enterprise-grade usages of SOAP messaging will typically specify critical information regarding the Sender, Recipient, Message ID, and Correlation information. The latter is needed to relate response messages to previous requests.

JAXM implementations may choose to support a number of industry standard messaging Profiles. Profiles are identified by name. A Profile name uniquely identifies a particular industry/standards body's usage of SOAP messaging. A JAXM Client must use the URI of the schema associated with a given Profile as a Profile name.

For Example the Profile for ebXML TR&P 1.0 would be identified by the the following URI:

http://www.ebxml.org/project_teams/transport/messageHeader.xsd

JAXM developers are required to specify, either at run-time or at deployment time, critical "system" level information necessary to correctly route, deliver and correlate messages. The way in which this information is mapped on to a given message is Profile specific. JAXM makes no assumptions about where this information, if present, is stored within a message. An explicit contract must therefore exist between a JAXM Client and it's Provider. The Profile String is used to establish this contract at run-time. In order for JAXM applications to be able to exchange business messages with peer-entities, an a-priori agreement (to use a common Profile) must exist. The way in which such agreements can be established is outside the scope of this document.

By way of example, an ebXML MS V1.0 Profile clearly specifies how a SOAP Header should be populated with necessary addressing, and message identification information. A JAXM Application, when using such a Profile, is responsible for constructing a SOAP Header as per the specifications associated with this Profile. All Providers supporting the same Profile (identified by a Profile name) will therefore have a common understanding of the message structure and message semantics. Note that JAXM Clients may specify Profile names when creating MessageFactory objects. Such objects can subsequently be used to produce Profile-specific Message Objects. In addition, JAXM implementations may choose to pre-populate a Message with critical information e.g. To & From in a profile specific manner.

If an application chooses not to specify a standard Profile, the JAXM Provider must default to using an application specific Profile. In such cases, JAXM developers can not be assured of any level of interoperability based on public standards. It is conceivable that a given Provider may support multiple application specific i.e. private Profiles.

JAXM.2.4 JAXM Deployment

JAXM Clients may be deployed in Servlet 2.3 containers and/or J2EE™1.3 Containers. The configuration details of the JAXM provider is beyond the scope of this document.

JAXM developers must specify two key pieces of information as part of the deployment process in a Servlet 2.3 Container:

<!ELEMENT client-endpoint (#PCDATA)>
<!--
The client-endpoint identifies a specific address for a JAXM client. Parties wishing to exchange messages must have a mutual understanding of and be aware of this address. The client-endpoint must be a URI.
-->
<!ELEMENT listener-url (#PCDATA)>
<!--
The listener-url element is the URL of the MessageListener.
-->

Note that the way in which the client configuration information is communicated to the Provider is an implementation detail.

In addition to the JAXM configuration information mentioned above, JAXM Clients are required to (if deploying a Client into a Servlet 2.3 Container) specify deployment information for JAXMServlets. The deployment descriptors for JAXMServlets will be specified in a future revision of this specification.

JAXM.2.5 MessageListener

JAXM promotes a standard way of delivering messages asynchronously to JAXM clients. In the case of J2EE™ EJB Containers, the `MessageListener` interface may be implemented using J2EE™ Message Driven Beans(MDB). In the case of Servlet Containers, JAXM clients are required to extend the JAXMServlet interface and implement the `onMessage()` method. As mentioned previously, the Provider to Client contract is based on SOAP1.1 (w/attachments) message formats. In other words, irrespective of where a JAXM client is deployed a standard message enveloping scheme is used. Furthermore, given that the `JAXM-Servlet` interface extends `HTTPServlet`, there is a clear assumption that when a JAXM Client is deployed in a Servlet Container the asynchronous activation model is built on SOAP1.1 (w/attachments) bound to HTTP.

Note that in a subsequent revision of this specification, and in the interest of JAXM Client portability, additional requirements may be introduced to more narrowly define the capabilities of the JAXMServlet.

JAXM.2.6 Message Security

JAXM introduces no new Security requirements. Messages are assumed to have both a transitory as well as a persistent confidentiality requirement. Support of security features and capabilities assuring confidentiality while messages are in transit are an implementation detail of the JAXM Provider. Where HTTP is the transport of choice, support for protocols such as SSL may be appropriate and adequate. In the case of SMTP infrastructures JAXM Providers may choose to use PGP and or S/MIME.

JAXM provides no specific interfaces to digital signatures that span an entire Message. The assumption is that JAXM developers will have access to “user” level portions of a Message - where “user” level is defined as the application specific parts of a message. Note that the signing or encrypting of the SOAP Header in a manner that would prevent the message from being interpreted and therefore correctly routed will raise a JAXM exception. A JAXM developer may require some application specific and potentially non-standard, encryption algorithms and/or security functions, to be applied to predefined portions of a message. In such circumstances, developers must select an appropriate Profile known to the JAXM provider.

JAXM developers may choose to use digital signature technologies to sign application level XML fragments as they see fit. As mentioned earlier, the application of specific signing technology must not interfere with the routing of Messages by the JAXM infrastructure.

The authentication of JAXM clients to JAXM providers is considered to be an implementation detail and beyond the scope of this specification.

CHAPTER JAXM.3

Package Overview

The following sections provide an overview of the JAXM packages. In addition, JAXM code samples of have been included. Note that the examples are not exhaustive and should not be considered a normative part of this specification. The intent is to provide an overview of the packages, the details of which are provided in subsequent sections of this document.

JAXM.3.1 Class hierarchy diagrams

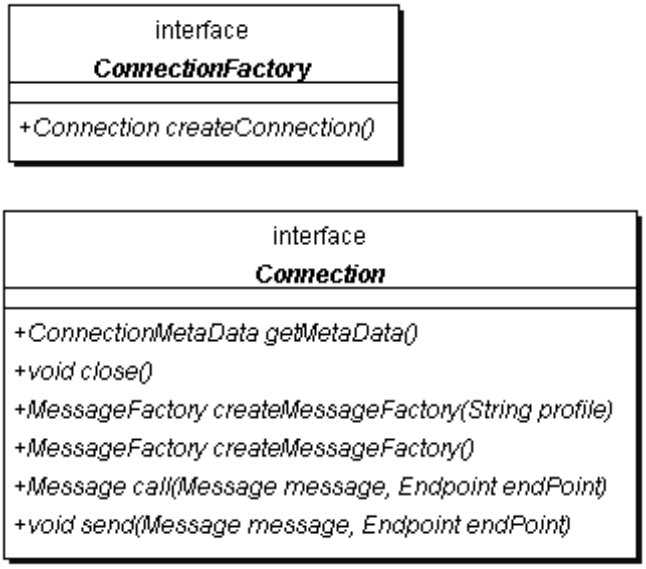

Figure 9 javax.xml.messaging Package

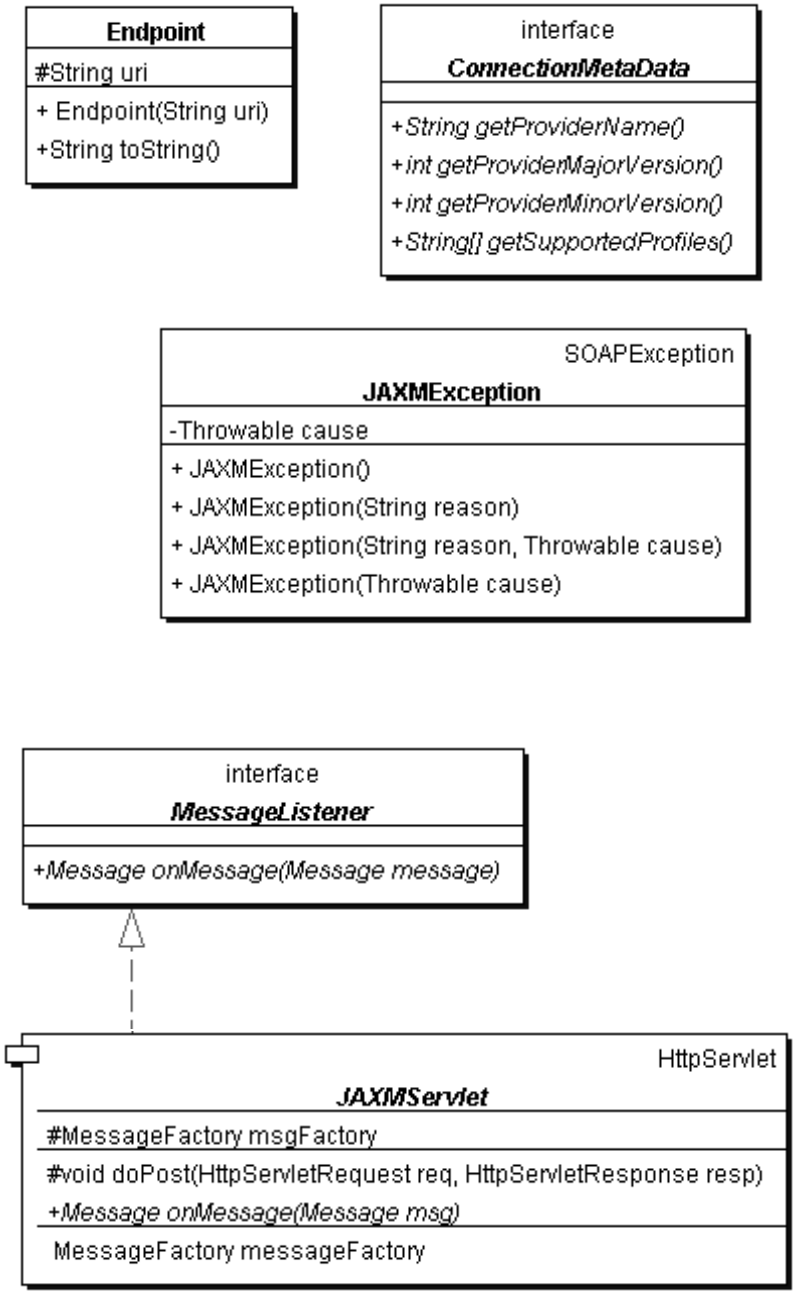

Figure 10 javax.xml.messaging Package continued.

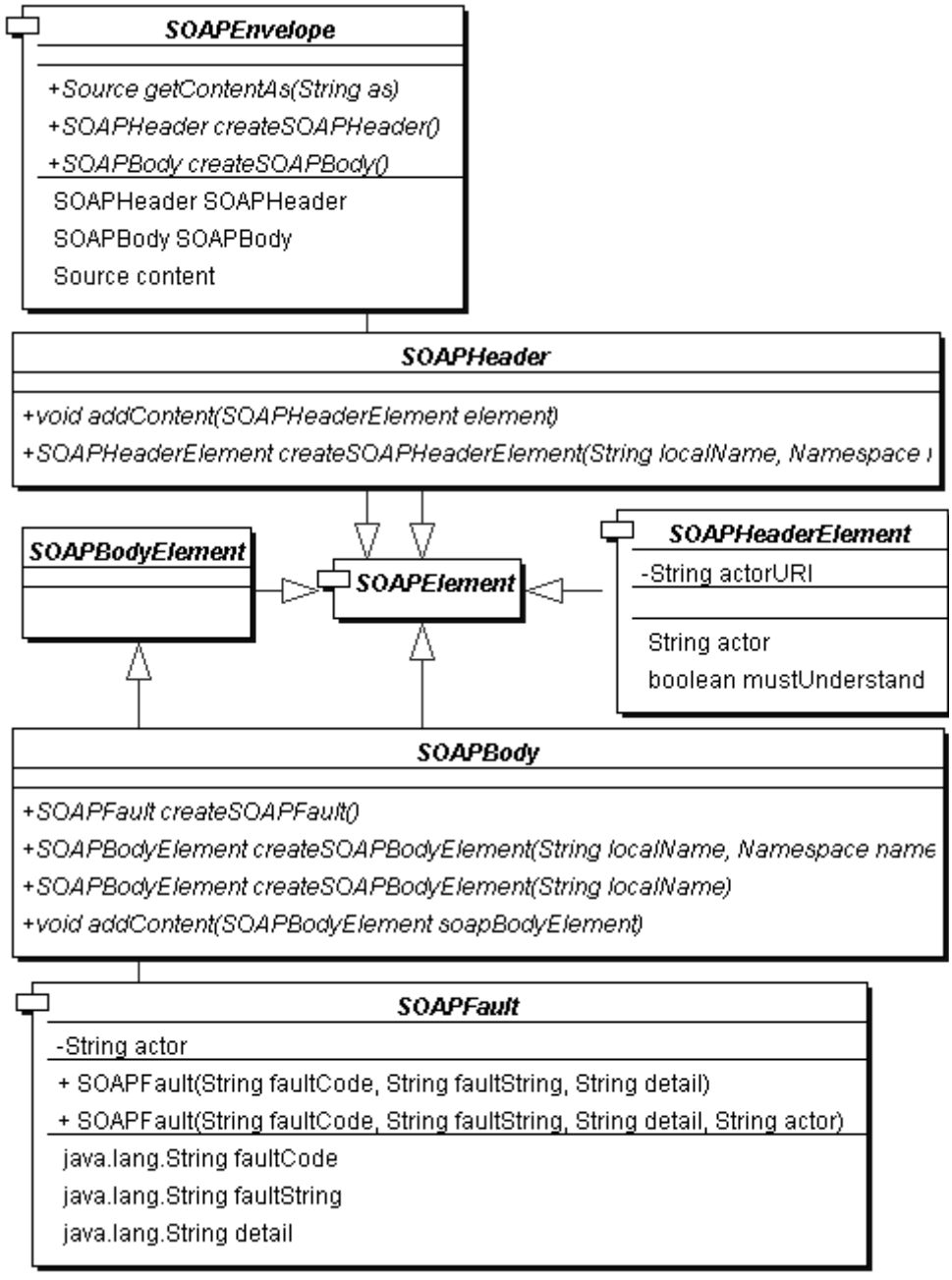

Figure 11 javax.xml.soap Package.

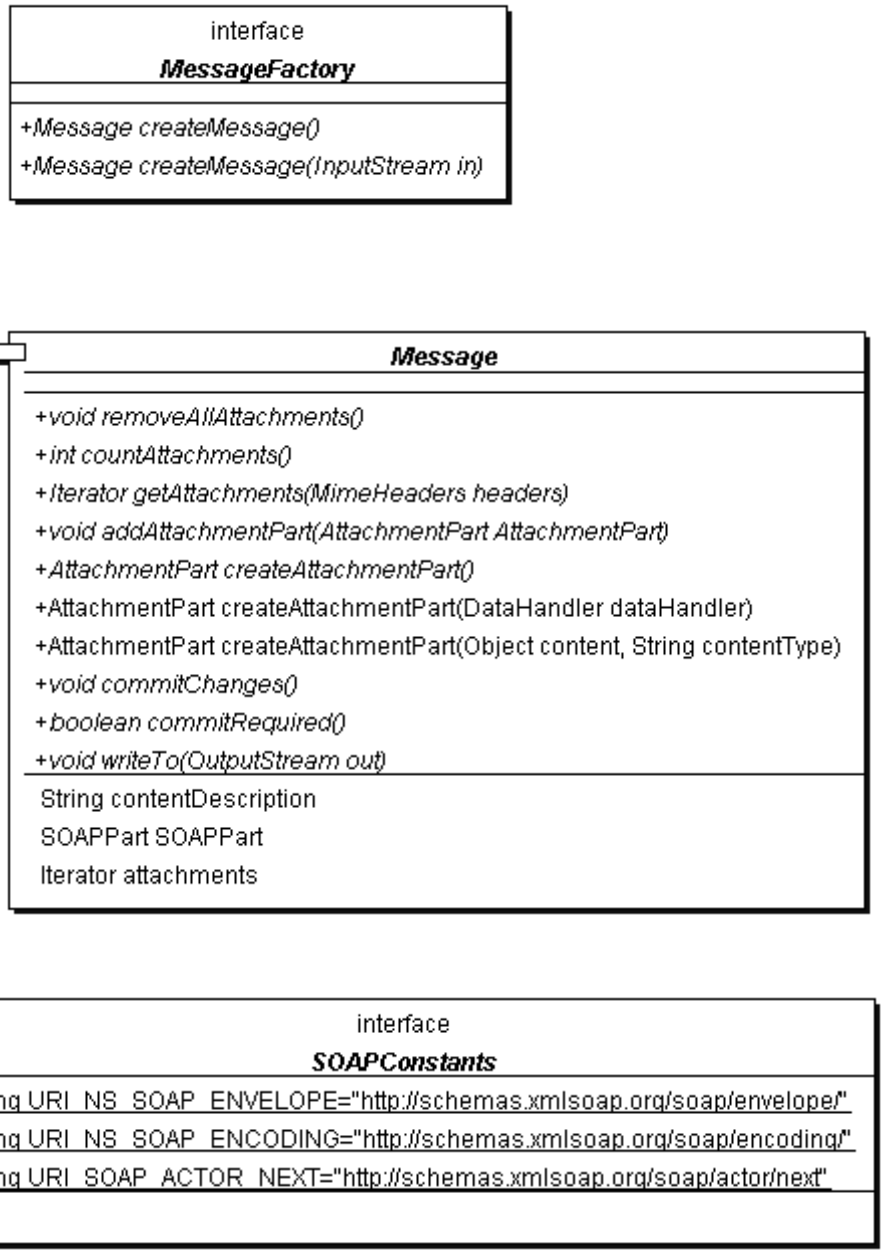

Figure 12 javax.xml.soap Package continued.

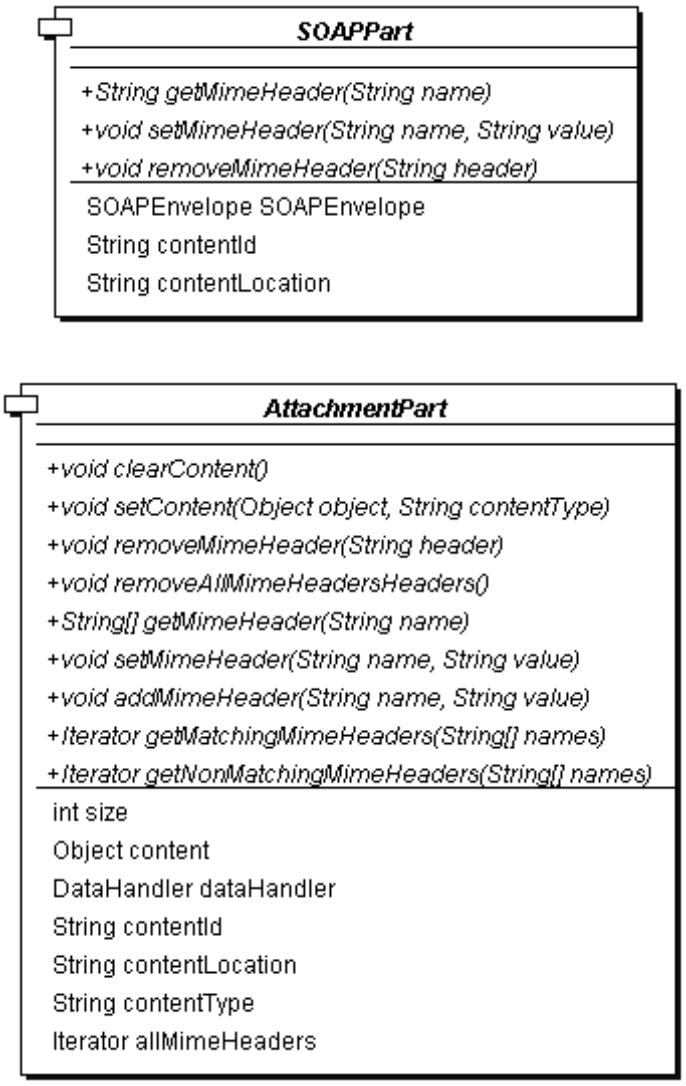

Figure 13 javax.xml.soap Package continued.

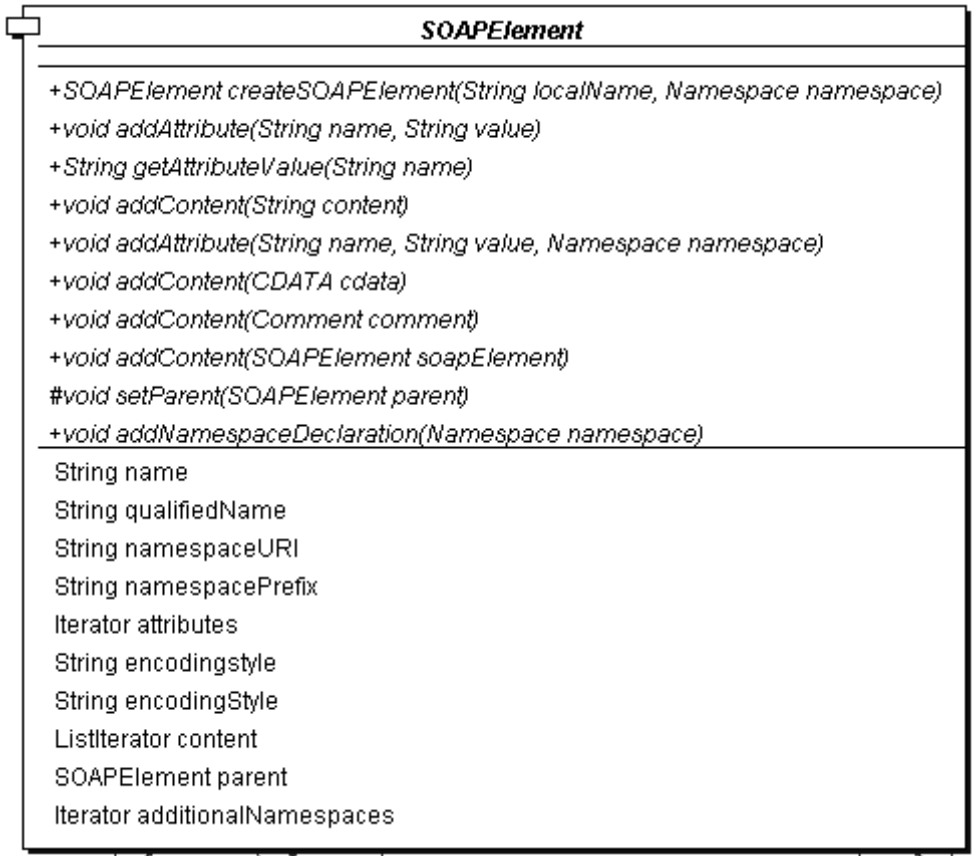

Figure 14 javax.xml.soap Package continued.

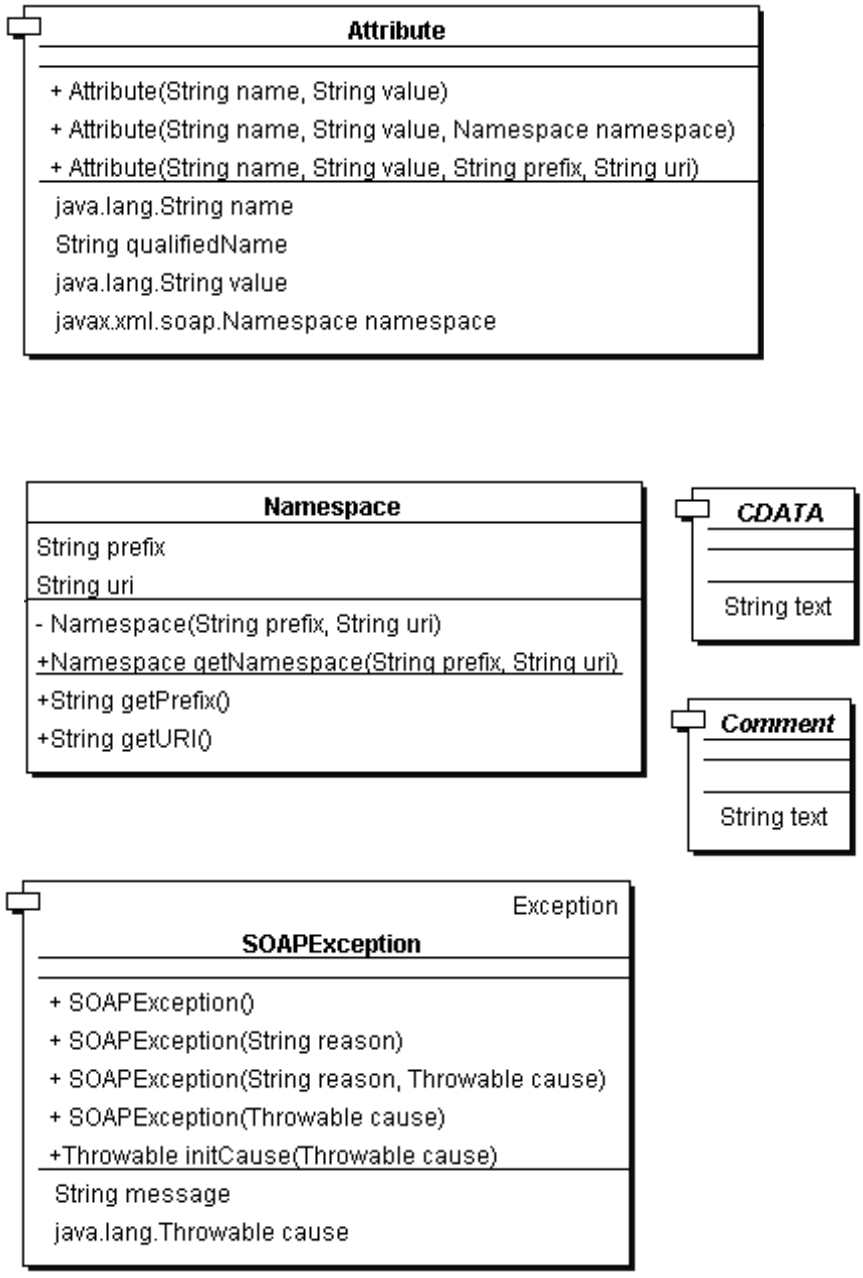

Figure 15 javax.xml.soap Package continued.

JAXM.3.2 Message Consumer example (no attachments)

The following Servlet based example demonstrates the usage of JAXM in a case where SOAP messages (without attachments) are consumed.

```
/*
* Copyright 2001 by Sun Microsystems, Inc.,
* 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
* All rights reserved.
*
* This software is the confidential and proprietary information
* of Sun Microsystems, Inc. ("Confidential Information"). You
* shall not disclose such Confidential Information and shall use
* it only in accordance with the terms of the license agreement
* you entered into with Sun.
*/
package com.sun.xml.messaging.examples;
import javax.xml.messaging.*;
import javax.xml.soap.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.activation.DataHandler;
import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class SOAPListener extends JAXMServlet {
    public void onMessage(Message message) {
        System.out.println(" onMessage of MyListener invoked");
        try {
            //Get the first part which is the SOAPPart.
            SOAPPart soapPart = message.getSOAPPart();
            SOAPEnvelope soapEnvelope = soapPart.getSOAPEnvelope();
            DOMSource domSrc = (DOMSource)
                   soapEnvelope.getContentAs(DOMSource.FEATURE);

            //Now use the dom tree to actually get traverse and
            //get some of the fields from it.
            Document doc = (Document)domSrc.getNode();
            Element root = doc.getDocumentElement();
            NodeList list =
            root.getElementsByTagName("GetLastTradePriceDetailed");
            Element element;
```

```
            Element childElement;
            for(int i =0; i < list.getLength(); i++) {
                if (!(list.item(i) instanceof Element)) {
                    continue;
                }
                element = (Element)list.item(i);
                NodeList list2 = element.getChildNodes();
                for(int j = 0; j < list2.getLength(); j++) {
                    if(!(list2.item(j) instanceof Element)) {
                        continue;
                    }
                    childElement = (Element)list2.item(j);
                    System.out.println(childElement.getTagName());
                    System.out.println("\t");
                    System.out.println(
                    ((Text)childElement.getFirstChild()).getData());
                    System.out.println("\n");
                }
            }
        } catch(Exception jxe) {
            jxe.printStackTrace();
        }
    }
}
```

JAXM.3.3 Message Producer example (no attachments)

The following Servlet based example demonstrates the usage of JAXM in a case where SOAP messages have no attachments.

```
/*
* Copyright 2001 by Sun Microsystems, Inc.,
* 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
* All rights reserved.
*
* This software is the confidential and proprietary information
* of Sun Microsystems, Inc. ("Confidential Information"). You
* shall not disclose such Confidential Information and shall use
* it only in accordance with the terms of the license agreement
* you entered into with Sun.
*/
package com.sun.xml.messaging.examples;
```

```
import javax.xml.messaging.*;
import javax.xml.soap.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.parsers.*;

import javax.servlet.http.*;
import java.io.IOException;
import org.w3c.dom.Document;

public class SOAPServlet extends HttpServlet {
    ConnectionFactory connectionFactory;

    public void init(ServletConfig servletConfig) throws
                                        ServletException {

    //Get the servlet context to lookup a connection factory for a
    //particular client id. When done in a standalone application or
    //even in servlets a more sophisticated mechanism like JNDI could
    //be used to lookup connection factories.
    ServletContext ctxt = servletConfig.getServletContext();

    //Lookup connection factory for Client1 and store it to create
    //connection to the provider to send messages.
    connectionFactory =
        (ConnectionFactory)ctxt.getAttribute("Client1");
    }

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException
    {
        try {
        //A request comes in to send a message. So we create a
        //connection from the factory.
            Connection connection =
                connectionFactory.createConnection();

        //Create a message factory from the Connection to produce
        //messages.
            MessageFactory mf = connection.createMessageFactory();

        //Create messgages from the message factory.
            Message message = mf.createMessage();
```

```
            //Get the SOAPPart from the message.
            //Note the infrastructure takes care of creating the
            //SOAPPart.
            //The user needs to get the SOAPPart and get the
            //SOAPEnvelope and populate it appropriately.
                SOAPPart soapPart = message.getSOAPPart();

            //Get the SOAPEnvelope from the header container obtained
            //above.
                SOAPEnvelope soapEnvelope = soapPart.getSOAPEnvelope();

            //Create a SOAPHeader from the SOAPEnvelope
                SOAPHeader soapHeader = soapEnvelope.createSOAPHeader();

            //Create a SOAPHeaderElement that will be appended to the
            //SOAPHeader.
                SOAPHeaderElement she =
                          soapHeader.createSOAPHeaderElement();
                she.setName("from", "http://foo.bar/", "m");
                she.addContent("foo@bar.com");
                soapHeader.addContent(she);
                soapEnvelope.setSOAPHeader(soapHeader);

            //Similarly create SOAPBody with SOAPBodyElements and append
            //that to the soapEnvelope..
                SOAPBody soapBody = soapEnvelope.createSOAPBody();
                SOAPBodyElement sbe = soapBody.createSOAPBodyElement();
                sbe.setName("purchaseOrder", "http://foo.bar.com",
                                                          "po");
                sbe.addContent("purchase order for a chair");
                soapEnvelope.setSOAPBody(soapBody);

                Endpoint endPoint =
                          new Endpoint("http://foo.bar/Service");
                connection.send(message, endPoint);
                response.setStatus(HttpServletResponse.SC_OK);
            } catch(Exception jxe) {
                jxe.printStackTrace();
            }
        }
    }
```

JAXM.3.4 Servlet based Message producer example

The following Servlet based example demonstrates the usage of JAXM in a case where SOAP messages with attachments are produced and sent to a business partner.

```
/*
* Copyright 2001 by Sun Microsystems, Inc.,
* 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
* All rights reserved.
*
* This software is the confidential and proprietary information
* of Sun Microsystems, Inc. ("Confidential Information"). You
* shall not disclose such Confidential Information and shall use
* it only in accordance with the terms of the license agreement
* you entered into with Sun.
*/
package com.sun.xml.messaging.examples;
import javax.xml.messaging.*;
import javax.xml.soap.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.activation.DataHandler;
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.IOException;
import java.net.URL;
import org.w3c.dom.Document;

public class SampleServlet extends HttpServlet {
    ConnectionFactory connectionFactory;

    public void init(ServletConfig servletConfig) throws
                                        ServletException
    {
        //Get the servlet context to lookup a connection factory for a
        //particular client id. When done in a standalone application
        //or even in servlets a more sophisticated mechanism like
        //JNDI could be used to lookup connection factories.
            ServletContext ctxt = servletConfig.getServletContext();

        //Lookup connection factory for Client1 and store it to
        //create connection to the provider to send messages.
```

```
        connectionFactory =
            (ConnectionFactory)ctxt.getAttribute("Client1");
    }

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException
    {
        try {
            //A request comes in to send a message. So we create
            //a connection from the factory.
            Connection connection =
                      connectionFactory.createConnection();

            //Create a message factory from the Connection to
            //produce messages.
            MessageFactory mf =
                      connection.createMessageFactory();

            //Create messgages from the message factory.
            Message message = mf.createMessage();

            //Get the SOAPPart from the message.
            //Note the infrastructure takes care of creating the
            //SOAPPart. The user needs to get the SOAPPart and
            //get the SOAPEnvelope and populate it appropriately.
            SOAPPart soapPart = message.getSOAPPart();

            //Get the SOAPEnvelope from the header container
            //obtained above.
            SOAPEnvelope soapEnvelope =
                        soapPart.getSOAPEnvelope();
            //Create a DOMSource.
            DOMSource domsrc = null;
            try {
                DocumentBuilderFactory dbf =
                    DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document doc =
                    db.parse("file:///foo.bar/soap.xml");
                domsrc = new DOMSource(doc);
            } catch(Exception e) {
                System.err.println("Error in creating DOMSource" +
                e.getMessage());
```

```
            }
            //Use the DOMSource created above to set the contents of
            //the SOAPEnvelope.
            soapEnvelope.setContent(domsrc);

            //Create an attachment and add it to the message.
            URL url = new URL("http://foo.bar/img.jpg");
            DataHandler dh = new DataHandler(url);
            AttachmentPart ap = message.createAttachmentPart(dh);
            message.addAttachmentPart(ap);

            //Create an Endpoint passing it the URI of the client with
            //which this client wants to exchange messages.
            Endpoint endPoint =
                       new Endpoint("http://foo.bar/Service");
            connection.send(message, endPoint);
            response.setStatus(HttpServletResponse.SC_OK);
        } catch(Exception jxe) {
            jxe.printStackTrace();
        }
    }
}
```

JAXM.3.5 Servlet based Message consumer example

The following Servlet based example demonstartes the usage of JAXM in a case where SOAP messages with attachments are received from a business partner and processed.

```
/*
* Copyright 2001 by Sun Microsystems, Inc.,
* 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
* All rights reserved.
*
* This software is the confidential and proprietary information
* of Sun Microsystems, Inc. ("Confidential Information"). You
* shall not disclose such Confidential Information and shall use
* it only in accordance with the terms of the license agreement
* you entered into with Sun.
*/
package com.sun.xml.messaging.examples;
import javax.xml.messaging.*;
```

```
import javax.xml.soap.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.activation.DataHandler;
import java.io.InputStream;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.xml.sax.SAXException;

public class MyListener extends JAXMServlet {
    Connection connection;
    public void init(Connection connection) {
        this.connection = connection;
    }

    public void destroy() {
    }

    public void onMessage(Message message) {
        System.out.println(" onMessage of MyListener invoked");
        try {
            //Get the first part which is the SOAPPart.
            SOAPPart soapPart = message.getSOAPPart();
            SOAPEnvelope soapEnvelope = soapPart.getSOAPEnvelope();
            DOMSource domSrc = (DOMSource)
                soapEnvelope.getContentAs(DOMSource.FEATURE);

            //Now use the dom tree to actually get traverse
            //and get some of the fields from it.

            Node node = domSrc.getNode();
            //Now start manipulating the attachments

            int count = message.countAttachments();
            System.out.println("No. of attachments " + count);
            AttachmentPart ap = null;
            for(int i = 0; i < count; i++) {
                ap = message.getAttachmentPart(i);
                if(ap.getDataHandler().getContentType()
                                .equals("text/xml")) {
                    StreamSource streamSource = (StreamSource)
                                ap.getContent();
                    DocumentBuilderFactory dbf =
```

```
                DocumentBuilderFactory.newInstance();
                dbf.setValidating(true);
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document doc =
                        db.parse(streamSource.getInputStream());
            }
          }
        } catch(JAXMException jxe) {
          jxe.printStackTrace();
        } catch(IOException ioe) {
          ioe.printStackTrace();
        } catch(ParserConfigurationException pce) {
          pce.printStackTrace();
        } catch(SAXException se) {
          se.printStackTrace();
        } catch(SOAPException spe) {
          spe.printStackTrace();
        }
    }
```

CHAPTER JAXM.4

Package javax.xml.messaging

Class Summary	
Interfaces	
`Connection` 34	A client's active connection to its messaging provider.
`ConnectionFactory` 38	A factory for creating connections to a particular messaging provider.
`ConnectionMetaData` 39	Information about the messaging provider to which a client has a connection.
`MessageListener` 48	The `MessageListener` interface serves as a marker interface for components (for example, servlets) that are intended to be consumers of JAXM messages.
Classes	
`Endpoint` 41	An opaque representation of an application endpoint.
`JAXMServlet` 45	`JAXMServlet` is the super-class for components that live in a servlet container that receive JAXM messages.
Exceptions	
`JAXMException` 43	An exception that signals that a JAXM exception has occurred.

javax.xml.messaging
Connection

Declaration

public interface `Connection`

Description

A client's active connection to its messaging provider.

A `Connection` object is created using a `ConnectionFactory` object, which is configured so that the connections it creates will be to a particular messaging provider. To create a connection, a client first needs to obtain an instance of the `ConnectionFactory` class that creates connections to the desired messaging provider. The client then calls the `createConnection` method on it to get a `Connection` object that connects the client with its messaging provider.

The information necessary to set up a `ConnectionFactory` that creates connections to a particular messaging provider is provided at deployment time. The client supplies its messaging service as well as a logical name, and the two are bound together in a naming service. Later the client can do a lookup on the logical name to retrieve an instance of the `ConnectionFactory` class that produces connections to its messaging provider.

The following code fragment is an example of a client doing a lookup of a `ConnectionFactory` object and then using it to create a connection. The first two lines in this example use the Java™ Naming and Directory Interface (JNDI) to create a context, which is then used to do the lookup. The argument provided to the `lookup` method is the logical name that was previously associated with the desired messaging provider. The `lookup` method returns a Java `Object`, which needs to be cast to a `ConnectionFactory` object before it can be used to create a connection. In the following code fragment, the resulting `Connection` object is a connection to the messaging provider that is associated with the logical name "ProviderXYZ".

```
Context ctx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)ctx.lookup("ProviderXYZ");
Connection con = cf.createConnection();
```

After the client has obtained a connection to its messaging provider, it can use that connection to create one or more `MessageFactory` objects. `Messages` can be created using the `MessageFactory` object. Finally, the message is delivered to an `Endpoint` using the `call()` or `send()` methods.

The messaging provider maintains a list of `Endpoint` objects, which is established at deployment time as part of configuring the messaging provider. A client can send messages only to those parties represented by `Endpoint` objects in its messaging provider's list.

Typically, because clients have one messaging provider, they will do all their messaging with a single `Connection` object. It is possible, however, for a sophisticated application to use multiple connections.

Generally, a container is configured with a `MessageListener` object at deployment time using an implementation-specific mechanism. A client running in such a container will use the `MessageListener` object to receive messages, which means that the client can receive messages asynchronously.

By contrast, a client that is running as a standalone application, and therefore has no `MessageListener` object, is unable to send and receive messages asynchronously. If a client wants a response to a message, it must send the message using the `Message` method `call()`. This method will send the message and block until it

gets a response, which it then returns to the client that sent the original message. In this synchronous messaging scenario, the client cannot send another message until after the response is returned.

Due to the authentication and communication setup done when a `Connection` object is created, it is a relatively heavy-weight object. Therefore, a client should close its connection as soon as it is done using it.

JAXM objects created using one `Connection` object cannot be used with a different `Connection` object.

Member Summary

Methods

public Message	`call(Message, Endpoint)`[35] Send the `Message` object and block until the provider receives a reply to this message from the recipient and return the reply message.
public void	`close()`[35] Closes this `Connection` object, freeing its resources and making it immediately available for garbage collection.
public MessageFactory	`createMessageFactory()`[36] Create a message factory assuming SOAP Messaging without profiles.
public MessageFactory	`createMessageFactory(String)`[36] Creates a `MessageFactory` object that will produce `Message` objects for a certain profile.
public ConnectionMetaData	`getMetaData()`[36] Retrieves the `ConnectionMetaData` object that contains information about the messaging provider to which this `Connection` object is connected.
public void	`send(Message, Endpoint)`[37] Sends the `Message` object.

Methods

call(Message, Endpoint)

public `Message`[65] call(`Message`[65] message, `Endpoint`[41] endPoint)
throws JAXMException

Send the `Message` object and block until the provider receives a reply to this message from the recipient and return the reply message.

Returns: the reply message.

Throws:

`JAXMException`[43] - if a JAXM transmission error occurs.

`UnsupportedOperationException` - if a JAXM provider can't support a synchronous request-reply operation.

close()

public void close()

throws JAXMException

Closes this `Connection` object, freeing its resources and making it immediately available for garbage collection. Since a provider typically allocates significant resources outside the JVM on behalf of a connection, clients should close connections when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Throws:
`JAXMException`$_{43}$ - if a JAXM error occurs while closing the connection.

createMessageFactory()

public `MessageFactory`$_{70}$ createMessageFactory()
throws JAXMException

Create a message factory assuming SOAP Messaging without profiles. This message factory will produce messages intended for the specified endpoint.

Parameters:
`receiver` - the endpoint for the destination.

Returns: a new `MessageFactory` object that will create `Message` objects

Throws:
`JAXMException`$_{43}$ - if the JAXM infrastructure encounters an error

createMessageFactory(String)

public `MessageFactory`$_{70}$ createMessageFactory(java.lang.String profile)
throws JAXMException

Creates a `MessageFactory` object that will produce `Message` objects for a certain profile. The `MessageFactory` that is returned could create instances of `Message` subclasses as appropriate for particular profiles.

Parameters:
`profile` - is a string that represents a particular JAXM profile in use. An example of a JAXM profile is: “ebxml”.

Returns: a new `MessageFactory` object that will create `Message` objects

Throws:
`JAXMException`$_{43}$ - if the JAXM infrastructure encounters an error, if for example the endpoint that is being used is not compatible with the specified profile.

getMetaData()

public `ConnectionMetaData`$_{39}$ getMetaData()
throws JAXMException

Retrieves the `ConnectionMetaData` object that contains information about the messaging provider to which this `Connection` object is connected.

Returns: the `ConnectionMetaData` object with information about the messaging provider

Throws:
`JAXMException`[43]

See Also: `ConnectionMetaData`[39]

send(Message, Endpoint)

public void send(`Message`[65] message, `Endpoint`[41] endPoint)
throws JAXMException

Sends the `Message` object. The message is sent to the specified endpoint.

This method returns immediately after handing the message over to the messaging provider. No assumptions can be made regarding the ultimate success or failure of message delivery at the time this method returns.

Throws:
`JAXMException`[43] - if a JAXM transmission error occurs.

See Also: `call(Message, Endpoint)`[35]

javax.xml.messaging
ConnectionFactory

Declaration

public interface `ConnectionFactory`

Description

A factory for creating connections to a particular messaging provider. A `ConnectionFactory` object is an administered object that is created by the container (a servlet or Enterprise JavaBeans™ container) in which an application runs.

A `ConnectionFactory` object is configured in an implementation- specific way, and the connections it creates will be to a particular messaging provider. A `ConnectionFactory` object is registered with a naming service, such as one based on Java Naming and Directory Interface™ (JNDI) technology. This registration associates the `ConnectionFactory` object with a logical name. When an application wants to establish a connection with the provider associated with that `ConnectionFactory` object, it does a lookup, providing the logical name. The application can then use the `ConnectionFactory` object that is returned to create a connection to the messaging provider.

Member Summary	
Methods	
`public Connection`	`createConnection()`[38] Creates a `Connection` object to the messaging provider that is the default provider for this `ConnectionFactory` object.

Methods

createConnection()

public `Connection`[34] createConnection() throws JAXMException

Creates a `Connection` object to the messaging provider that is the default provider for this `ConnectionFactory` object.

Returns: a `Connection` object that represents a connection to the default provider for this `ConnectionFactory` object

Throws:

`JAXMException`[43] - if there is an error in creating the connection

javax.xml.messaging
ConnectionMetaData

Declaration

public interface `ConnectionMetaData`

Description

Information about the messaging provider to which a client has a connection.

After obtaining a connection to its messaging provider, a client can get information about that provider. The following code fragment demonstrates how the `Connection` object `con` can be used to retrieve its `ConnectionMetaData` object and then to get the name and version number of the messaging provider.

```
ConnectionMetaData cmd = con.getMetaData();
String name = cmd.getProviderName();
int majorVersion = cmd.getProviderMajorVersion();
int minorVersion = cmd.getProviderMinorVersion();
```

Member Summary

Methods

`public int`	`getProviderMajorVersion()` *39* Retrieves an `int` indicating the major version number of the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.
`public int`	`getProviderMinorVersion()` *40* Retrieves an `int` indicating the minor version number of the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.
`public String`	`getProviderName()` *40* Retrieves a `String` containing the name of the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.
`public String`	`getSupportedProfiles()` *40* Retrieves a list of the messaging profiles that are supported by the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.

Methods

getProviderMajorVersion()

public int getProviderMajorVersion()

Retrieves an `int` indicating the major version number of the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.

Returns: the messaging provider's major version number as an `int`

getProviderMinorVersion()

public int getProviderMinorVersion()

Retrieves an `int` indicating the minor version number of the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.

Returns: the messaging provider's minor version number as an `int`

getProviderName()

public java.lang.String getProviderName()

Retrieves a `String` containing the name of the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected. This string is provider implementation dependent. It could either describe a particular instance of the provider or just the name of the provider.

Returns: the messaging provider's name as a `String`

getSupportedProfiles()

public java.lang.String[] getSupportedProfiles()

Retrieves a list of the messaging profiles that are supported by the messaging provider to which the `Connection` object described by this `ConnectionMetaData` object is connected.

Returns: a `String` array in which each element is a messaging profile supported by this provider.

javax.xml.messaging
Endpoint

Declaration

public class **Endpoint**

```
java.lang.Object
  |
  +--javax.xml.messaging.Endpoint
```

Description

An opaque representation of an application endpoint. Typically, an `Endpoint` object represents a business entity, but it may represent a party of any sort. Conceptually, an `Endpoint` object is the mapping of a logical name (example, a URI) to a physical location, such as a URL.

A client supplies an `Endpoint` object to the `Connection` methods that create a `MessageProducer` object to tell the message producer where the messages it creates should go.

The default identification of an Endpoint is a URI. The provider needs to be configured using a deployment-specific mechanism with mappings from an `Endpoint` to the physical details of that end point.

`Endpoint` objects can also be looked up in a naming service. This scheme is more flexible because logical identifiers can be bound and rebound to specific URIs or even other naming schemes (such as DUNS numbers).

The default `Endpoint` as defined in JAXM supports URI-based identification of parties. This defines what JAXM providers need to support at minimum for identification of destinations.

Member Summary

Fields

`protected`	`uri` *42*

Constructors

`public`	`Endpoint(String)` *42* Construct an `Endpoint` object using a URI.

Methods

`public String`	`toString()` *42* Retrieves a string representation of this `Endpoint` object.

Inherited Member Summary

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

uri

Fields

uri

protected java.lang.String uri

Constructors

Endpoint(String)

public Endpoint(java.lang.String uri)

Construct an `Endpoint` object using a URI.

Parameters:

`uri` - the URI that identifies a party.

Methods

toString()

public java.lang.String toString()

Retrieves a string representation of this `Endpoint` object. This string is likely to be provider-specific, and programmers are discouraged from parsing and programmatically interpreting the contents of this string.

Overrides: java.lang.Object.toString() in class java.lang.Object

Returns: a `String` with a provider-specific representation of this `Endpoint` object

javax.xml.messaging
JAXMException

Declaration

public class `JAXMException` extends `SOAPException`[95]

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--javax.xml.soap.SOAPException[95]
                    |
                    +--javax.xml.messaging.JAXMException
```

All Implemented Interfaces: java.io.Serializable

Description

An exception that signals that a JAXM exception has occurred. A `JAXMException` object may contain a `String` that gives the reason for the exception, an embedded `Throwable` object, or both. This class provides methods for retrieving reason messages and for retrieving the embedded `Throwable` object.

Typical reasons for throwing a `JAXMException` object are problems such as, not being able to send a message, and not being able to get a connection with the provider. Reasons for embedding a `Throwable` object include problems such as an input/output errors or a parsing problem, such as an error parsing a header.

Member Summary

Constructors

public	`JAXMException()`[44] Constructs a `JAXMException` object with no reason or embedded `Throwable` object.
public	`JAXMException(String)`[44] Constructs a `JAXMException` object with the given `String` as the reason for the exception being thrown.
public	`JAXMException(String, Throwable)`[44] Constructs a `JAXMException` object with the given `String` as the reason for the exception being thrown and the given `Throwable` object as an embedded exception.
public	`JAXMException(Throwable)`[44] Constructs a `JAXMException` object initialized with the given `Throwable` object.

Inherited Member Summary

Methods inherited from class java.lang.Object

Inherited Member Summary
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`
Methods inherited from interface `SOAPException`[95]
`getCause()`[97], `getMessage()`[97], `initCause(Throwable)`[97]
Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, printStackTrace, printStackTrace, printStackTrace, toString`

Constructors

JAXMException()

public JAXMException()

Constructs a `JAXMException` object with no reason or embedded `Throwable` object.

JAXMException(String)

public JAXMException(java.lang.String reason)

Constructs a `JAXMException` object with the given `String` as the reason for the exception being thrown.

Parameters:

`reason` - a description of what caused the exception

JAXMException(String, Throwable)

public JAXMException(java.lang.String reason, java.lang.Throwable cause)

Constructs a `JAXMException` object with the given `String` as the reason for the exception being thrown and the given `Throwable` object as an embedded exception.

Parameters:

`reason` - a description of what caused the exception

`cause` - a `Throwable` object that is to be embedded in this `JAXMException` object

JAXMException(Throwable)

public JAXMException(java.lang.Throwable cause)

Constructs a `JAXMException` object initialized with the given `Throwable` object.

javax.xml.messaging
JAXMServlet

Declaration

public abstract class `JAXMServlet` extends javax.servlet.http.HttpServlet implements `MessageListener`[48]

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--javax.xml.messaging.JAXMServlet
```

All Implemented Interfaces: `MessageListener`[48], java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

Description

`JAXMServlet` is the super-class for components that live in a servlet container that receive JAXM messages.

A `JAXMServlet` is notified of a message arrival using the HTTP-SOAP binding.

Member Summary

Fields

protected	`msgFactory`[46]

Constructors

public	`JAXMServlet()`[46]

Methods

protected void	`doPost(HttpServletRequest, HttpServletResponse)`[46] This doPost(..) implementation will internalize the SOAP message that was posted to this servlet and call onMessage with the internalized message.
public abstract Message	`onMessage(Message)`[47]
public void	`setMessageFactory(MessageFactory)`[47] A `MessageFactory` for a particular profile needs to be set before a message can be received and successfully internalized.

Inherited Member Summary

Methods inherited from class javax.servlet.GenericServlet

Inherited Member Summary
`destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log`
Methods inherited from class javax.servlet.http.HttpServlet
`doDelete, doGet, doOptions, doPut, doTrace, getLastModified, service, service`
Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Fields

msgFactory

protected `MessageFactory`$_{70}$ msgFactory

Constructors

JAXMServlet()

public JAXMServlet()

Methods

doPost(HttpServletRequest, HttpServletResponse)

protected void doPost(javax.servlet.http.HttpServletRequest req,
javax.servlet.http.HttpServletResponse resp)
throws ServletException, IOException

This doPost(..) implementation will internalize the SOAP message that was posted to this servlet and call onMessage with the internalized message. Note that the msgFactory instance variable will be used to internalize the message. This ensures that the message factory for the correct profile is used to internalize the message.

Overrides: javax.servlet.http.HttpServlet.doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse) in class javax.servlet.http.HttpServlet

Throws:
`IOException`, `ServletException`

onMessage(Message)

public abstract `Message`[65] onMessage(`Message`[65] msg)

Specified By: `onMessage(Message)`[48] in interface `MessageListener`[48]

setMessageFactory(MessageFactory)

public void setMessageFactory(`MessageFactory`[70] msgFactory)

A `MessageFactory` for a particular profile needs to be set before a message can be received and successfully internalized.

javax.xml.messaging
MessageListener

Declaration

public interface `MessageListener`

All Known Implementing Classes: `JAXMServlet`[45]

Description

The `MessageListener` interface serves as a marker interface for components (for example, servlets) that are intended to be consumers of JAXM messages.

See Also: `JAXMServlet`[45]

Member Summary	
Methods	
`public Message`	`onMessage(Message)`[48] Passes the given `Message` object to this `MessageListener` object.

Methods

onMessage(Message)

public `Message`[65] onMessage(`Message`[65] message)

Passes the given `Message` object to this `MessageListener` object. This method is invoked by the messaging provider. It is expected that EJB Containers will deliver JAXM Messages to EJB components using MDBs that implement `javax.xml.messaging.MessageListener`.

Parameters:
`message` - the `Message` object to be passed to this `MessageListener` object

Returns: the response. If this is null, then the original message is treated as a "oneway" message.

CHAPTER JAXM.5

Package javax.xml.soap

Class Summary

Interfaces

`MessageFactory` 70	A factory for creating `Message` objects.
`SOAPConstants` 84	This interface defines a number of constants pertaining to the SOAP 1.1 protocol.

Classes

`AttachmentPart` 51	A single attachment of a SOAP `Message` object, which may contain multiple `AttachmentPart` objects.
`Attribute` 58	This class represents an xml attribute that can occur anywhere in a SOAP Message.
`CDATA` 61	This class represents an xml CDATA that can occur anywhere in a SOAP Message.
`Comment` 63	This class represents an xml comment that can occur anywhere in a SOAP Message.
`Message` 65	The root class for all SOAP messages.
`MimeHeader` 72	`MimeHeader` stores a MIME header name and its value.
`MimeHeaders` 74	`MimeHeaders` is a container for all MIME headers present in a MIME part.
`Namespace` 77	This class represents an xml attribute that can occur anywhere in a SOAP Message.
`SOAPBody` 79	A SOAP body object represents the contents of the SOAP body element.
`SOAPBodyElement` 82	A SOAPBodyElement object represents the contents in the SOAPBody.
`SOAPElement` 85	A SOAPElement object represents the contents in a SOAPBody, the contents in a SOAPHeaderElement, the content that can follow the SOAPBody in a SOAPEnvelope or what can follow the detail element in a SOAPFault.
`SOAPEnvelope` 91	The container for the SOAPHeader and SOAPBody portion of a `SOAPPart` object.
`SOAPFault` 99	SOAPFault carries error and/or status information within a SOAP message.
`SOAPHeader` 102	A SOAP header object represents the contents of the SOAP header element.
`SOAPHeaderElement` 104	A SOAPHeaderElement object represents the contents in the SOAPHeader.
`SOAPPart` 107	The container for the SOAP specific portion of a `Message` object.

Exceptions

Class Summary	
SOAPException95	An exception that signals that a SOAP exception has occurred.

javax.xml.soap
AttachmentPart

Declaration

public abstract class **AttachmentPart**

```
java.lang.Object
  |
  +--javax.xml.soap.AttachmentPart
```

Description

A single attachment of a SOAP `Message` object, which may contain multiple `AttachmentPart` objects. Each `AttachmentPart` object consists of two parts, a content and an associated header. The latter consists of name/value pairs that identify and describe the content.

An `AttachmentPart` object must conform to certain standards.

1. It must conform to MIME [RFC2045] standards (http://www.ietf.org/rfc/rfc2045.txt)
2. It MUST contain content
3. The header portion MUST include the following headers:

- `Content-Type`
  Content-Type identifies the type of data in the content of a `AttachmentPart` object and MUST conform to [RFC2045]. The following is an example of a Content-Type header:

```
Content-Type:  application/xml
```

The following line of code, in which `ap` is an `AttachmentPart` object, sets the header in the example.

```
ap.setMimeHeader("Content-Type", "application/xml");
```

There are no restrictions on the content portion of an `AttachmentPart` object, they may be anything from a simple-plain-text-object to a complex XML document.

An `AttachmentPart` object is created with the method `Message.createAttachmentPart`. After setting the MIME headers in the `AttachmentPart`, it is added to the message that created it with the method `Message.addAttachmentPart`.

The following code fragment, in which `m` is a `Message` object and `contentStringl` is a `String`, creates an instance of `AttachmentPart`, sets the `AttachmentPart` object with some content and header information, and adds the `AttachmentPart` object to the `Message` object.

```
AttachmentPart ap1 = m.createAttachmentPart();
ap1.setContent(contentString1, "text/plain");
m.addAttachmentPart(ap1);
```

For "text/plain", "text/html" and "text/xml" MIME content-types, the `DataContentHandler` does the conversions to and from Java types corresponding to the MIME types. For other MIME types, you can pass an `InputStream` that contains the content data.

The following code fragment creates and adds a second `AttachmentPart` instance to the same message. buff is a binary byte buffer representing the jpeg.

```
AttachmentPart ap2 = m.createAttachmentPart();
byte[] jpegData =  ...;
ap2.setContent(new ByteArrayInputStream(jpegData), "image/jpeg");
m.addAttachmentPart(ap2);
```

To retrieve the contents and header from an `AttachmentPart`, the `getContent()` method can be used. Depending on the `DataContentHandlers` present the returned `Object` can either be a typed Java object corresponding to the MIME type or an `InputStream` which contains the contents as bytes. The method `clearContent` removes all the content from an `AttachmentPart` object but does not affect its header information.

```
bp1.clearContent();
```

Member Summary

Constructors

public	`AttachmentPart()` 53

Methods

public abstract void	`addMimeHeader(String, String)` 53 Add a MIME header with the specified name and value to this attachment.
public abstract void	`clearContent()` 53 Clears out the content of this `AttachmentPart` object.
public abstract Iterator	`getAllMimeHeaders()` 54 Return all the headers as an Iterator over MimeHeader objects
public abstract Object	`getContent()` 54 Get the content of this `AttachmentPart` as a Java object.
public String	`getContentId()` 54 Convienience method to get the value of the MIME header Content-Id.
public String	`getContentLocation()` 54 Convienience method to get the value of the MIME header Content-Location.
public String	`getContentType()` 54 Convienience method to get the value of the MIME header Content-Type.
public abstract DataHandler	`getDataHandler()` 54 Get the `DataHandler`.
public abstract Iterator	`getMatchingMimeHeaders(String[])` 55 Return all matching MimeHeader objects
public abstract String	`getMimeHeader(String)` 55 Gets all the values of the header that is identified by the given `String`.
public abstract Iterator	`getNonMatchingMimeHeaders(String[])` 55 Return all non-matching MimeHeader objects
public abstract int	`getSize()` 55 Returns the number of bytes in this `AttachmentPart` object.
public abstract void	`removeAllMimeHeadersHeaders()` 55 Remove all the MIME header entries.
public abstract void	`removeMimeHeader(String)` 55 Remove all MIME headers that match the given name.
public abstract void	`setContent(Object, String)` 56 Set the content of this attachment part.
public void	`setContentId(String)` 56 Convienience method to set the value of the MIME header Content-Id.
public void	`setContentLocation(String)` 56 Convienience method to set the value of the MIME header Content-Location.
public void	`setContentType(String)` 56 Convienience method to set the value of the MIME header Content-Type.
public abstract void	`setDataHandler(DataHandler)` 57 Set the `DataHandler`.

Member Summary	
public abstract void	setMimeHeader(String, String)[57] Change the first header entry that matches name to have value, adding a new header if no existing header matches.

Inherited Member Summary
Methods inherited from class java.lang.Object clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

AttachmentPart()

public AttachmentPart()

Methods

addMimeHeader(String, String)

public abstract void addMimeHeader(java.lang.String name, java.lang.String value)

Add a MIME header with the specified name and value to this attachment.

Note that RFC822 headers can only contain US-ASCII characters.

Parameters:

`name` - header name

`value` - header value

Throws:

`IllegalArgumentException` - if there was a problem with the specified mime header name and value.

clearContent()

public abstract void clearContent()

Clears out the content of this `AttachmentPart` object. The MIME header portion is left untouched.

getAllMimeHeaders()

public abstract java.util.Iterator getAllMimeHeaders()

Return all the headers as an Iterator over MimeHeader objects

Returns: MimeHeader objects

getContent()

public abstract java.lang.Object getContent()
throws SOAPException

Get the content of this `AttachmentPart` as a Java object. The type of the returned Java object depends on the `DataContentHandler` that is used to interpret the bytes and the `Content-Type`. A JAXM-compliant provider must, as a minimum, return `java.lang.String` corresponding to any content stream with a `Content-Type` value of `text/plain` and a `javax.xml.transform.StreamSource` corresponding to a content stream with a `Content-Type` value of `text/xml`. For those content types that an installed `DataHandler` does not understand, the `DataHandler` is required to return `java.io.InputStream` with the raw bytes.

Returns: a Java object

Throws:

`SOAPException`$_{95}$ - if there is no content set into this `AttachmentPart` or if there was a data transformation error.

getContentId()

public java.lang.String getContentId()

Convienience method to get the value of the MIME header Content-Id.

getContentLocation()

public java.lang.String getContentLocation()

Convienience method to get the value of the MIME header Content-Location.

getContentType()

public java.lang.String getContentType()

Convienience method to get the value of the MIME header Content-Type.

getDataHandler()

public abstract javax.activation.DataHandler getDataHandler()
throws SOAPException

Get the `DataHandler`.

Returns: the `DataHandler` associated with this `AttachmentPart`.

Throws:

a - `SOAPException` if there is no data in this `AttachmentPart` object.

`SOAPException`[95]

getMatchingMimeHeaders(String[])

public abstract java.util.Iterator getMatchingMimeHeaders(java.lang.String[] names)

Return all matching MimeHeader objects

Returns: matching MimeHeader objects

getMimeHeader(String)

public abstract java.lang.String[] getMimeHeader(java.lang.String name)

Gets all the values of the header that is identified by the given `String`.

Parameters:

`name` - the name of the header; example: “Content-Type”

Returns: a `String` giving the value for the specified header

See Also: `setMimeHeader(String, String)`[57]

getNonMatchingMimeHeaders(String[])

public abstract java.util.Iterator getNonMatchingMimeHeaders(java.lang.String[] names)

Return all non-matching MimeHeader objects

Returns: non-matching MimeHeader objects

getSize()

public abstract int getSize()

Returns the number of bytes in this `AttachmentPart` object.

Returns: the size of this `AttachmentPart` object in bytes or -1 if the size cannot be determined

removeAllMimeHeadersHeaders()

public abstract void removeAllMimeHeadersHeaders()

Remove all the MIME header entries.

removeMimeHeader(String)

public abstract void removeMimeHeader(java.lang.String header)

Remove all MIME headers that match the given name.

Parameters:
`header` - the string name of the MIME header/s that is to be removed.

setContent(Object, String)

public abstract void setContent(java.lang.Object object, java.lang.String contentType)

Set the content of this attachment part. The type of the `object` should correspond to the `contentType` value. This depends on the particular set of `DataContentHandlers` in use.

Parameters:
`object` - the Java object which makes up the content for this attachment part.

`contentType` - the MIME string that specifies the type of this content.

Throws:
`IllegalArgumentException` - if the contentType does not match the type of the content object, or if there was no `DataContentHandler` for this content object.

See Also: `getContent()` *54*

setContentId(String)

public void setContentId(java.lang.String contentId)

Convienience method to set the value of the MIME header Content-Id.

Throws:
`IllegalArgumentException` - if there was a problem with the specified `contentId`.

setContentLocation(String)

public void setContentLocation(java.lang.String contentLocation)

Convienience method to set the value of the MIME header Content-Location.

Throws:
`IllegalArgumentException` - if there was a problem with the specified `content location`.

setContentType(String)

public void setContentType(java.lang.String contentType)

Convienience method to set the value of the MIME header Content-Type.

Throws:
`IllegalArgumentException` - if there was a problem with the specified `content type`.

setDataHandler(DataHandler)

public abstract void setDataHandler(javax.activation.DataHandler dataHandler)

Set the `DataHandler`. Typically, on an incoming message the `DataHandler` is automatically set. When creating a message and populating the message with content, the `setDataHandler` method can be used to get data from various data sources into this message.

Parameters:

`the` - `DataHandler` object.

Throws:

`IllegalArgumentException` - if there was a problem with the specified `dataHandler`.

setMimeHeader(String, String)

public abstract void setMimeHeader(java.lang.String name, java.lang.String value)

Change the first header entry that matches name to have value, adding a new header if no existing header matches. Remove all matching headers but the first.

Note that RFC822 headers can only contain US-ASCII characters

Parameters:

`name` - header name

`value` - header value

Throws:

`IllegalArgumentException` - if there was a problem with the specified mime header name and value.

javax.xml.soap
Attribute

Declaration

public class `Attribute`

```
java.lang.Object
  |
  +--javax.xml.soap.Attribute
```

Description

This class represents an xml attribute that can occur anywhere in a SOAP Message.

Member Summary

Constructors

public	`Attribute(String, String)` 59 Create an attribute with the specified name and value.
public	`Attribute(String, String, Namespace)` 59 Create an attribute with the specified name, value and namespace
public	`Attribute(String, String, String, String)` 59 Create an attribute with the specified name, value namespace prefix and namespace uri

Methods

public String	`getName()` 59 Returns the local name of the attribute
public Namespace	`getNamespace()` 60 Return the namespace associated with the attribute.
public String	`getQualifiedName()` 60 Returns the fully qualified name of the attribute
public String	`getValue()` 60 Returns the value associated with this attribute

Inherited Member Summary

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

Attribute(String, String)

public Attribute(java.lang.String name, java.lang.String value)

Create an attribute with the specified name and value.

Parameters:

`String` - name of the attribute.

`String` - value of the attribute.

Attribute(String, String, Namespace)

public Attribute(java.lang.String name, java.lang.String value, `Namespace`$_{77}$ namespace)

Create an attribute with the specified name, value and namespace

Parameters:

`String` - name of the attribute.

`String` - value of the attribute.

`Namespace` - namespace associated with the attribute.

Attribute(String, String, String, String)

public Attribute(java.lang.String name, java.lang.String value, java.lang.String prefix, java.lang.String uri)

Create an attribute with the specified name, value namespace prefix and namespace uri

Parameters:

`String` - name of the attribute.

`String` - value of the attribute.

`String` - namespace prefix associated with the attribute.

`String` - namespace uri associated with the attribute.

Methods

getName()

public java.lang.String getName()

Returns the local name of the attribute

Returns: String the local name.

getNamespace()

public Namespace[77] getNamespace()

Return the namespace associated with the attribute.

Returns: Namespace the namespace associated with the attribute.

getQualifiedName()

public java.lang.String getQualifiedName()

Returns the fully qualified name of the attribute

Returns: String the fully qualified name.

getValue()

public java.lang.String getValue()

Returns the value associated with this attribute

Returns: String the attribute value.

javax.xml.soap
CDATA

Declaration

public class **CDATA**

```
java.lang.Object
  |
  +--javax.xml.soap.CDATA
```

Description

This class represents an xml CDATA that can occur anywhere in a SOAP Message.

Member Summary

Constructors

public	CDATA() 62

Methods

public String	getText() 62 Return the text of this cdata.
public void	setText(String) 62 Set the content of the CDATA.

Inherited Member Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

CDATA()

public CDATA()

Methods

getText()

public java.lang.String getText()

Return the text of this cdata.

Returns: String the cdata.

setText(String)

public void setText(java.lang.String text)

Set the content of the CDATA.

Parameters:

`String` - the text of this cdata.

javax.xml.soap
Comment

Declaration

public class **Comment**

```
java.lang.Object
  |
  +--javax.xml.soap.Comment
```

Description

This class represents an xml comment that can occur anywhere in a SOAP Message.

Member Summary

Constructors

public	Comment()[64]

Methods

public String	getText()[64] Return the text of this comment.
public void	setText(String)[64] Set the content of the Comment.

Inherited Member Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Comment()

public Comment()

Methods

getText()

public java.lang.String getText()

Return the text of this comment.

Returns: String the comment.

setText(String)

public void setText(java.lang.String text)

Set the content of the Comment.

Parameters:

`String` - the text of this comment.

javax.xml.soap
Message

Declaration

public abstract class `Message`

```
java.lang.Object
  |
  +--javax.xml.soap.Message
```

Description

The root class for all SOAP messages.

A `Message` object consists of a SOAP part and optionally one or more attachment part(s). The SOAP part for a `Message` object is a `SOAPPart` object, which contains information used for message routing and identification. The SOAP part can be retrieved by calling the method `Message.getSOAPPart()`. As transmitted on the "wire", a header is an XML document.

An attachment may or may not be an XML document. It consists of zero or more `AttachmentPart` objects, each of which contain application specific data. The `Message` interface provides methods for creating `AttachmentPart` objects and also for adding them to a `Message` object. A party that has received a `Message` object can examine its contents by retrieving individual attachment parts.

Member Summary

Constructors

public	`Message()` 66

Methods

public abstract void	`addAttachmentPart(AttachmentPart)` 66 Adds the given `AttachmentPart` object to this `Message` object.
public abstract void	`commitChanges()` 67 `commitChanges()` updates this message with all the changes made to it.
public abstract boolean	`commitRequired()` 67 Indicates whether the message has been committed.
public abstract int	`countAttachments()` 67 Get a count of the number of attachments in this message.
public abstract AttachmentPart	`createAttachmentPart()` 67 Creates a new empty `AttachmentPart` object.
public AttachmentPart	`createAttachmentPart(DataHandler)` 67 A convenience method to create an `AttachmentPart` and populate it using a `DataHandler`.
public AttachmentPart	`createAttachmentPart(Object, String)` 68 A convenience method to create a `AttachmentPart` and populate it with the specified data and content type.
public abstract Iterator	`getAttachments()` 68 Get all the `AttachmentPart` objects that are part of this Message.
public abstract Iterator	`getAttachments(MimeHeaders)` 68 Get all the `AttachmentPart` objects that have header entries that match the specified headers.

Member Summary	
public abstract String	getContentDescription()[68] Get a label describing this message.
public abstract SOAP-Part	getSOAPPart()[68] Get the SOAP part of this Message object.
public abstract void	removeAllAttachments()[68] Removes all AttachmentPart objects that have been added to this Message object.
public abstract void	setContentDescription(String)[69] Set a label describing this message.
public abstract void	writeTo(OutputStream)[69] Writes this Message object to the given output stream.

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Message()

public Message()

Methods

addAttachmentPart(AttachmentPart)

public abstract void addAttachmentPart(AttachmentPart[51] AttachmentPart)

Adds the given AttachmentPart object to this Message object. A AttachmentPart must be created before adding it to a message.

Parameters:

AttachmentPart - an AttachmentPart object that is to become part of this Message object's payload.

Throws:

IllegalArgumentException, IllegalStateException

commitChanges()

public abstract void commitChanges() throws SOAPException

`commitChanges()` updates this message with all the changes made to it. `commitChanges` is automatically called before a `send()`, `call()`, or `writeTo`. However, if changes are made to a message that was received or to one that was already sent, then `commitChanges()` saves these changes and generates any changes that can be read back (for example: a MessageId in profiles that support a message id).

`commitChanges()` marks the point at which the data from all constituent `AttachmentPart` objects are pulled into the message.

Throws:
`SOAPException`[95]

commitRequired()

public abstract boolean commitRequired()

Indicates whether the message has been committed.

Returns: boolean `true` if `commitChanges`message has been called atleast once on this message, `false` otherwise.

countAttachments()

public abstract int countAttachments()

Get a count of the number of attachments in this message. This count does not include the SOAP part.

Returns: integer count of the number of attachments.

createAttachmentPart()

public abstract `AttachmentPart`[51] createAttachmentPart()

Creates a new empty `AttachmentPart` object. Note that the method `addAttachmentPart` must be called with this new `AttachmentPart` object as the parameter in order for it to become an attchament to this `Message` object.

Returns: a new `AttachmentPart` object that can be added to this `Message` object as an attachment

createAttachmentPart(DataHandler)

public `AttachmentPart`[51] createAttachmentPart(javax.activation.DataHandler dataHandler)

A convenience method to create an `AttachmentPart` and populate it using a `DataHandler`.

Returns: an attachment part which contains the data generated from the specified `DataHandler`.

Throws:
`IllegalArgumentException` - if there was a problem with the specified `dataHandler`.

createAttachmentPart(Object, String)

public AttachmentPart[51] createAttachmentPart(java.lang.Object content, java.lang.String contentType)

A convenience method to create a AttachmentPart and populate it with the specified data and content type.

Returns: an attachment part which contains the specified data.

Throws:

IllegalArgumentException - if the contentType does not match the type of the content object, or if there was no DataContentHandler for this content object.

getAttachments()

public abstract java.util.Iterator getAttachments()

Get all the AttachmentPart objects that are part of this Message.

Returns: an iterator over all the attachments of this message.

getAttachments(MimeHeaders)

public abstract java.util.Iterator getAttachments(MimeHeaders[74] headers)

Get all the AttachmentPart objects that have header entries that match the specified headers. Note that a returned attachment could have headers in addition to those specified.

Returns: an iterator over all matching attachments.

getContentDescription()

public abstract java.lang.String getContentDescription()

Get a label describing this message.

Returns: a string label describing this message, or null.

getSOAPPart()

public abstract SOAPPart[107] getSOAPPart()

Get the SOAP part of this Message object. A message object need not have multiple attachment parts. Where a Message object contains one or more attachments the SOAP Part must be the first MIME body part in the message.

Returns: the SOAPPart object that serves as the header for this Message object

removeAllAttachments()

public abstract void removeAllAttachments()

Removes all AttachmentPart objects that have been added to this Message object.

This method does not touch the SOAP part.

setContentDescription(String)

public abstract void setContentDescription(java.lang.String description)

Set a label describing this message.

Parameters:
`description` - The description string

writeTo(OutputStream)

public abstract void writeTo(java.io.OutputStream out)
throws IOException

Writes this `Message object to the given output stream. The externalization format is as defined by the SOAP 1.1 with Attachments scheme.`

If there are no attachments, then just an XML stream is written out. For those messages that have attachments, writeTo `writes a MIME encoded byte stream.`

Parameters:
`out` - the `OutputStream` object to which this `Message` object will be written

Throws:
`IOException` - if a SOAP error occurs

javax.xml.soap
MessageFactory

Declaration

public interface `MessageFactory`

Description

A factory for creating `Message` objects.

A JAXM client performs the following steps to create a message.

- Creates a `MessageFactory` object from the connection.
```
MessageFactory mf = con.createMessageFactory();
```
- Calls the method `createMessage` on the `MessageFactory` object.
```
Message m = mf.createMessage();
```

Member Summary	
Methods	
`public Message`	`createMessage()` [70] Creates a new `Message` object with all the standard message header information.
`public Message`	`createMessage(InputStream)` [71] Internalize the contents of an `InputStream` into a `Message` object.

Methods

createMessage()

public `Message` [65] createMessage()
throws SOAPException

Creates a new `Message` object with all the standard message header information. This `Message` object can be sent "as is" when a message containing only a SOAP part is sufficient. Otherwise, the `Message` object needs to create one or more `AttachmentPart` objects and add them to itself to form the payload of the message.

Returns: a new `Message` object with standard header information and this `MessageFactory` object's default destination and expiration

Throws:

`SOAPException` [95] - if a JAXM error occurs

createMessage(InputStream)

public Message[65] createMessage(java.io.InputStream in)
throws IOException, SOAPException

Internalize the contents of an `InputStream` into a `Message` object.

Parameters:
`in` - the `InputStream` that contains the data for a message.

Throws:
`SOAPException`[95], `IOException`

javax.xml.soap
MimeHeader

Declaration

public class `MimeHeader`

```
java.lang.Object
  |
  +--javax.xml.soap.MimeHeader
```

Description

`MimeHeader` stores a MIME header name and its value.

Member Summary		
Constructors		
	public	`MimeHeader(String, String)` 72 Construct a Header object.
Methods		
	public String	`getName()` 73 Returns the name of this header.
	public String	`getValue()` 73 Returns the value of this header.

Inherited Member Summary
Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MimeHeader(String, String)

public MimeHeader(java.lang.String name, java.lang.String value)

Construct a Header object.

Parameters:

`name` - name of the header

`value` - value of the header

Methods

getName()

public java.lang.String getName()

Returns the name of this header.

Returns: name of the header

getValue()

public java.lang.String getValue()

Returns the value of this header.

Returns: value of the header

javax.xml.soap
MimeHeaders

Declaration

public class `MimeHeaders`

```
java.lang.Object
  |
  +--javax.xml.soap.MimeHeaders
```

Description

`MimeHeaders` is a container for all MIME headers present in a MIME part.

This class is primarily used when the JAXM application wants to retrieve specific attachments based on certain MIME headers and values (see Message.getAttachments). `AttachmentPart` and other MIME dependent parts of the JAXM API will most likely use this class in their implementations.

Member Summary

Constructors

public	`MimeHeaders()` 75

Methods

public void	`addHeader(String, String)` 75 Add a header with the specified name and value to the header list.
public Iterator	`getAllHeaders()` 75 Return all the headers as an Iterator over MimeHeader objects
public String	`getHeader(String)` 75 Return all the values for the specified header.
public Iterator	`getMatchingHeaders(String[])` 75 Return all matching MimeHeader objects
public Iterator	`getNonMatchingHeaders(String[])` 76 Return all non-matching MimeHeader objects
public void	`removeAllHeaders()` 76 Remove all the header entries.
public void	`removeHeader(String)` 76 Remove all header entries that match the given name
public void	`setHeader(String, String)` 76 Change the first header entry that matches name to have value, adding a new header if no existing header matches.

Inherited Member Summary

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

MimeHeaders()

public MimeHeaders()

Methods

addHeader(String, String)

public void addHeader(java.lang.String name, java.lang.String value)

Add a header with the specified name and value to the header list.

Note that RFC822 headers can only contain US-ASCII characters.

Parameters:

`name` - header name

`value` - header value

Throws:

`IllegalArgumentException` - if there was a problem in the mime header name or value being added.

getAllHeaders()

public java.util.Iterator getAllHeaders()

Return all the headers as an Iterator over MimeHeader objects

Returns: MimeHeader objects

getHeader(String)

public java.lang.String[] getHeader(java.lang.String name)

Return all the values for the specified header. The values are String objects.

Parameters:

`name` - header name

getMatchingHeaders(String[])

public java.util.Iterator getMatchingHeaders(java.lang.String[] names)

Return all matching MimeHeader objects

Returns: matching MimeHeader objects

getNonMatchingHeaders(String[])

public java.util.Iterator getNonMatchingHeaders(java.lang.String[] names)

Return all non-matching MimeHeader objects

Returns: non-matching MimeHeader objects

removeAllHeaders()

public void removeAllHeaders()

Remove all the header entries.

removeHeader(String)

public void removeHeader(java.lang.String name)

Remove all header entries that match the given name

Parameters:

`name` - header name

setHeader(String, String)

public void setHeader(java.lang.String name, java.lang.String value)

Change the first header entry that matches name to have value, adding a new header if no existing header matches. Remove all matching headers but the first.

Note that RFC822 headers can only contain US-ASCII characters

Parameters:

`name` - header name

`value` - header value

Throws:

`IllegalArgumentException` - if there was a problem in the mime header name or value being set.

javax.xml.soap
Namespace

Declaration

public class **Namespace**

```
java.lang.Object
  |
  +--javax.xml.soap.Namespace
```

Description
This class represents an xml attribute that can occur anywhere in a SOAP Message.

Member Summary		
Methods		
public static Namespace	getNamespace(String, String)[77]	This will return the namespace object if there exists one for the prefix and uri else it will create a new object for the specified prefix and uri and return that.
public String	getPrefix()[78]	Returns the prefix associated with this namespace object
public String	getURI()[78]	Returns the uri associated with this namespace object

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods

getNamespace(String, String)

public static Namespace[77] getNamespace(java.lang.String prefix, java.lang.String uri)

This will return the namespace object if there exists one for the prefix and uri else it will create a new object for the specified prefix and uri and return that.

Parameters:

prefix - the prefix to be associated with the namespace

`uri` - the uri associated with the namespace.

getPrefix()

public java.lang.String getPrefix()

Returns the prefix associated with this namespace object

Returns: String the prefix associated.

getURI()

public java.lang.String getURI()

Returns the uri associated with this namespace object

Returns: String the uri associated

javax.xml.soap
SOAPBody

Declaration

public abstract class **SOAPBody** extends **SOAPElement**[85]

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement[85]
        |
        +--javax.xml.soap.SOAPBody
```

Description

A SOAP body object represents the contents of the SOAP body element. A SOAP body element consists of XML data that affects the way the application specific content is processed by the message handler.

Member Summary

Constructors

public	SOAPBody()[80]

Methods

public abstract void	addContent(SOAPBodyElement)[80] Add a SOAPBodyElement to the SOAPHeader.
public abstract SOAPBodyElement	createSOAPBodyElement(String)[80] Creates a new SOAPBodyElement.
public abstract SOAPBodyElement	createSOAPBodyElement(String, Namespace)[80] Creates a new SOAPBodyElement.
public abstract SOAPFault	createSOAPFault()[80] Creates a new SOAPFault.

Inherited Member Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from class SOAPElement[85]

addAttribute(String, String, Namespace)[87], addAttribute(String, String, Namespace)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addNamespaceDeclaration(Namespace)[88], createSOAPElement(String, Namespace)[88], getAdditionalNamespaces()[88], getAttributeValue(String)[89], getAttributes()[88], getContent()[89], getEncodingStyle()[89], getName()[89], getNamespacePrefix()[89], getNamespaceURI()[89], getParent()[90], getQualifiedName()[90], setEncodingstyle(String)[90], setParent(SOAPElement)[90]

Constructors

SOAPBody()

public SOAPBody()

Methods

addContent(SOAPBodyElement)

public abstract void addContent(`SOAPBodyElement`$_{82}$ soapBodyElement)

Add a SOAPBodyElement to the SOAPHeader.

Parameters:
`SOAPBodyElement` - the SOAPBodyElement to be appended as a child to the SOAPBody.

Throws:
`IllegalArgumentException` - if there is a problem in the body element being added.

createSOAPBodyElement(String)

public abstract `SOAPBodyElement`$_{82}$ createSOAPBodyElement(java.lang.String localName)
throws SOAPException

Creates a new SOAPBodyElement.

Returns: SOAPBodyElement - the new SOAPBodyElement created.

Throws:
`SOAPException`$_{95}$

createSOAPBodyElement(String, Namespace)

public abstract `SOAPBodyElement`$_{82}$ createSOAPBodyElement(java.lang.String localName,
`Namespace`$_{77}$ namespace)
throws SOAPException

Creates a new SOAPBodyElement.

Returns: SOAPBodyElement - the new SOAPBodyElement created.

Throws:
`SOAPException`$_{95}$

createSOAPFault()

public abstract `SOAPFault`$_{99}$ createSOAPFault()

throws SOAPException

Creates a new SOAPFault.

Returns: SOAPFault - the new SOAPFault created.

Throws:

`SOAPException`[95]

javax.xml.soap
SOAPBodyElement

Declaration

public abstract class **SOAPBodyElement** extends **SOAPElement**$_{85}$

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement85
        |
        +--javax.xml.soap.SOAPBodyElement
```

Direct Known Subclasses: SOAPFault$_{99}$

Description

A SOAPBodyElement object represents the contents in the SOAPBody.

Member Summary	
Constructors	
public	SOAPBodyElement()$_{83}$

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Methods inherited from class SOAPElement$_{85}$
addAttribute(String, String, Namespace)$_{87}$, addAttribute(String, String, Namespace)$_{87}$, addContent(SOAPElement)$_{87}$, addContent(SOAPElement)$_{87}$, addContent(SOAPElement)$_{87}$, addContent(SOAPElement)$_{87}$, addNamespaceDeclaration(Namespace)$_{88}$, createSOAPElement(String, Namespace)$_{88}$, getAdditionalNamespaces()$_{88}$, getAttributeValue(String)$_{89}$, getAttributes()$_{88}$, getContent()$_{89}$, getEncodingStyle()$_{89}$, getName()$_{89}$, getNamespacePrefix()$_{89}$, getNamespaceURI()$_{89}$, getParent()$_{90}$, getQualifiedName()$_{90}$, setEncodingstyle(String)$_{90}$, setParent(SOAPElement)$_{90}$

Constructors

SOAPBodyElement()

public SOAPBodyElement()

javax.xml.soap
SOAPConstants

Declaration

public interface `SOAPConstants`

Description
This interface defines a number of constants pertaining to the SOAP 1.1 protocol.

Member Summary	
Fields	
`public static final`	`URI_NS_SOAP_ENCODING`[84] The namespace identifier for the SOAP encoding (see section 5 of the SOAP 1.1 specification).
`public static final`	`URI_NS_SOAP_ENVELOPE`[84] The namespace identifier for the SOAP envelope.
`public static final`	`URI_SOAP_ACTOR_NEXT`[84] The URI identifying the first application processing a SOAP request as the intended actor for a SOAP header entry (see section 4.2.2 of the SOAP 1.1 specification).

Fields

URI_NS_SOAP_ENCODING

public static final java.lang.String URI_NS_SOAP_ENCODING

The namespace identifier for the SOAP encoding (see section 5 of the SOAP 1.1 specification).

URI_NS_SOAP_ENVELOPE

public static final java.lang.String URI_NS_SOAP_ENVELOPE

The namespace identifier for the SOAP envelope.

URI_SOAP_ACTOR_NEXT

public static final java.lang.String URI_SOAP_ACTOR_NEXT

The URI identifying the first application processing a SOAP request as the intended actor for a SOAP header entry (see section 4.2.2 of the SOAP 1.1 specification).

javax.xml.soap
SOAPElement

Declaration

public abstract class **SOAPElement**

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement
```

Direct Known Subclasses: SOAPBody[79], SOAPBodyElement[82], SOAPEnvelope[91], SOAPHeader[102], SOAPHeaderElement[104]

Description

A SOAPElement object represents the contents in a SOAPBody, the contents in a SOAPHeaderElement, the content that can follow the SOAPBody in a SOAPEnvelope or what can follow the detail element in a SOAPFault. It is the base class for all of the classes that represent the SOAP objects as defined in the SOAP specification.

Member Summary

Constructors

public	SOAPElement()[86]

Methods

public abstract void	addAttribute(String, String)[86] Adds the attribute to the list of attributes for the element.
public abstract void	addAttribute(String, String, Namespace)[87] Adds the attribute to the list of attributes for the element.
public abstract void	addContent(CDATA)[87] Add a CDATA to the SOAPElement.
public abstract void	addContent(Comment)[87] Add a Comment to the SOAPElement.
public abstract void	addContent(SOAPElement)[87] Add a child SOAPElement to this SOAPElement.
public abstract void	addContent(String)[88] Add String content to the SOAPElement.
public abstract void	addNamespaceDeclaration(Namespace)[88] Add an additional namespace declaration to associate with this element.
public abstract SOAPElement	createSOAPElement(String, Namespace)[88] Creates a new SOAPElement.
public abstract Iterator	getAdditionalNamespaces()[88] Returns an iterator of the namespaces associated.
public abstract Iterator	getAttributes()[88] Returns the attributes for the element.
public abstract String	getAttributeValue(String)[89] Get the value of the attribute specified.
public abstract ListIterator	getContent()[89] Return the content of this Element.

Member Summary	
public abstract String	getEncodingStyle()89 returns the encoding style associated with the element.
public abstract String	getName()89 Returns the local name of the element.
public abstract String	getNamespacePrefix()89 Returns the prefix that is associated with the namespace.
public abstract String	getNamespaceURI()89 Returns the namespace URI associated with the element.
public abstract SOAPElement	getParent()90 Returns the parent of this element.
public abstract String	getQualifiedName()90 Returns the fully qualified name of the element.
public abstract void	setEncodingstyle(String)90 Set the encoding style for the element.
protected abstract void	setParent(SOAPElement)90 Sets the parent of this element.

Inherited Member Summary
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

SOAPElement()

public SOAPElement()

Methods

addAttribute(String, String)

public abstract void addAttribute(java.lang.String name, java.lang.String value)

Adds the attribute to the list of attributes for the element.

Parameters:

`name` - The name of the attribute.

`value` - The value of the attribute.

Throws:

`IllegalArgumentException` - if the specified attribute is illegal.

addAttribute(String, String, Namespace)

public abstract void addAttribute(java.lang.String name, java.lang.String value, `Namespace`[77] namespace)

Adds the attribute to the list of attributes for the element.

Parameters:

`String` - name The name of the attribute.

`String` - value The value of the attribute.

`Namespace` - The namespace of the attribute.

Throws:

`IllegalArgumentException` - if the specified attribute is illegal.

addContent(CDATA)

public abstract void addContent(`CDATA`[61] cdata)

Add a CDATA to the SOAPElement.

Parameters:

`cdata` - the CDATA to be added.

Throws:

`IllegalArgumentException` - if the content being set is invalid.

addContent(Comment)

public abstract void addContent(`Comment`[63] comment)

Add a Comment to the SOAPElement.

Parameters:

`comment` - the comment to be added.

Throws:

`IllegalArgumentException` - if the content being set is invalid.

addContent(SOAPElement)

public abstract void addContent(`SOAPElement`[85] soapElement)

Add a child SOAPElement to this SOAPElement.

Parameters:

`soapElement` - the SOAPElement to be added.

Throws:

`IllegalArgumentException` - if the content being set is invalid.

addContent(String)

public abstract void addContent(java.lang.String content)

Add String content to the SOAPElement.

Parameters:
`content` - the content to be added to the SOAPElement.

Throws:
`IllegalArgumentException` - if the content being set is invalid.

addNamespaceDeclaration(Namespace)

public abstract void addNamespaceDeclaration(`Namespace`$_{77}$ namespace)

Add an additional namespace declaration to associate with this element.

Parameters:
`namespace` - the namespace.

Throws:
`IllegalArgumentException` - If the namespace is already in use or if there is any other error in the namespace being set.

createSOAPElement(String, Namespace)

public abstract `SOAPElement`$_{85}$ createSOAPElement(java.lang.String localName, `Namespace`$_{77}$ namespace) throws SOAPException

Creates a new SOAPElement.

Parameters:
`localName` - The local part of the SOAPElement name.

`namespace` - The namespace associated with the element.

Returns: SOAPElement the new SOAPElement created.

Throws:
`SOAPException`$_{95}$ - if there is an error in creating the SOAPElement

getAdditionalNamespaces()

public abstract java.util.Iterator getAdditionalNamespaces()

Returns an iterator of the namespaces associated.

getAttributes()

public abstract java.util.Iterator getAttributes()

Returns the attributes for the element.

getAttributeValue(String)

public abstract java.lang.String getAttributeValue(java.lang.String name)

Get the value of the attribute specified.

Parameters:
`name` - The name of the attribute

Returns: The value associated with the particular attribute or null if there is no attribute with that name.

getContent()

public abstract java.util.ListIterator getContent()

Return the content of this Element. This `ListIterator` returned only contains all children —- both `SOAPElements` as well as `CDATA` and `Comments`.

Note that the `List` returned is an "active" list, which means that you can add `SOAPElement`, `CDATA` and `Comment` objects to this list.

Returns: java.util.List - A list of the contents of this SOAPElement

getEncodingStyle()

public abstract java.lang.String getEncodingStyle()

returns the encoding style associated with the element.

Returns: the encoding style.

getName()

public abstract java.lang.String getName()

Returns the local name of the element.

Returns: String - The local part of the name.

getNamespacePrefix()

public abstract java.lang.String getNamespacePrefix()

Returns the prefix that is associated with the namespace.

Returns: String - the prefix associated with the namespace.

getNamespaceURI()

public abstract java.lang.String getNamespaceURI()

Returns the namespace URI associated with the element.

Returns: String the namespace associated with the element.

getParent()

public abstract SOAPElement[85] getParent()

Returns the parent of this element.

Returns: SOAPElement the parent element.

getQualifiedName()

public abstract java.lang.String getQualifiedName()

Returns the fully qualified name of the element. A fully qualified name is represented as the "`prefix:localName`.

setEncodingstyle(String)

public abstract void setEncodingstyle(java.lang.String encodingStyle)

Set the encoding style for the element.

Parameters:
`encodingStyle` - the encoding style.

Throws:
`IllegalArgumenException` - if there was a problem in the encoding style being set.

setParent(SOAPElement)

protected abstract void setParent(SOAPElement[85] parent)

Sets the parent of this element.

Parameters:
`parent` - the parent element.

Throws:
`IllegalArgumentException` - If there is a problem in setting the parent.

javax.xml.soap
SOAPEnvelope

Declaration

public abstract class `SOAPEnvelope` extends `SOAPElement`[85]

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement[85]
        |
        +--javax.xml.soap.SOAPEnvelope
```

Description

The container for the SOAPHeader and SOAPBody portion of a `SOAPPart` object.

A client can access the SOAP Header and SOAP Body of a `SOAPEnvelope` object by calling the method `SOAPEnvelope.getSOAPHeader()` and `SOAPEnvelope.getSOAPBody()`. The following line of code, in which `se` is a `SOAPEnvelope` object, retrieves the SOAP Header and SOAP Body from the SOAP-Envelope.

```
SOAPHeader sh = se.getSOAPHeader();
SOAPBody sb = se.getSOAPBody();
```

To create a SOAPHeader and a SOAPBody a client can use the methods `SOAPEnvelope.create-SOAPHeader()` and `SOAPEnvelope.createSOAPBody()` and to set the SOAPHeader and SOAPBody on the SOAPEnvelope the client can use the methods `SOAPEnvelope.setSOAPHeader()` and `SOAPEnvelope.setSOAPBody()`. The following piece of code shows how to create and set the SOAP Header and SOAP Body on the SOAP Envelope.

```
SOAPHeader sh = se.createSOAPHeader();
SOAPBody sb = se.createSOAPBody();
//Add the necessary elements to the header and body and then set it
//on the SOAPEnvelope as follows:
se.setSOAPHeader(sh);
se.setSOAPBody(sb);
```

Member Summary

Constructors

public	`SOAPEnvelope()`[92]

Methods

public abstract SOAP-Body	`createSOAPBody()`[92] Creates a new SOAPBody.
public abstract SOAP-Header	`createSOAPHeader()`[93] Creates a new SOAPHeader.
public abstract Source	`getContentAs(String)`[93] Returns the content of the SOAPEnvelope as a source.
public abstract SOAP-Body	`getSOAPBody()`[93] Retruns the SOAPBody from the SOAPEnvelope.

Member Summary	
public abstract SOAPHeader	getSOAPHeader()[93] Retruns the SOAPHeader, if set, from the SOAPEnvelope.
public abstract void	setContent(Source)[93] Set the content of the SOAPEnvelope as a Source
public abstract void	setSOAPBody(SOAPBody)[93] Set the SOAPBody part of the envelope.
public abstract void	setSOAPHeader(SOAPHeader)[94] Set the SOAPHeader part of the envelope.

Inherited Member Summary
Methods inherited from class java.lang.Object clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Methods inherited from class SOAPElement[85] addAttribute(String, String, Namespace)[87], addAttribute(String, String, Namespace)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addNamespaceDeclaration(Namespace)[88], createSOAPElement(String, Namespace)[88], getAdditionalNamespaces()[88], getAttributeValue(String)[89], getAttributes()[88], getContent()[89], getEncodingStyle()[89], getName()[89], getNamespacePrefix()[89], getNamespaceURI()[89], getParent()[90], getQualifiedName()[90], setEncodingstyle(String)[90], setParent(SOAPElement)[90]

Constructors

SOAPEnvelope()

public SOAPEnvelope()

Methods

createSOAPBody()

public abstract SOAPBody[79] createSOAPBody() throws SOAPException

Creates a new SOAPBody.

Returns: SOAPBody - the new SOAPBody created.

Throws:
SOAPException[95]

createSOAPHeader()

public abstract SOAPHeader[102] createSOAPHeader()
throws SOAPException

Creates a new SOAPHeader.

Returns: SOAPHeader - the new SOAPHeader created.

Throws:
SOAPException[95]

getContentAs(String)

public abstract javax.xml.transform.Source getContentAs(java.lang.String as)

Returns the content of the SOAPEnvelope as a source.

Returns: javax.xml.transform.Source.

getSOAPBody()

public abstract SOAPBody[79] getSOAPBody()

Retruns the SOAPBody from the SOAPEnvelope.

Returns: SOAPBody the SOAPBody object associated with the SOAPEnvelope.

getSOAPHeader()

public abstract SOAPHeader[102] getSOAPHeader()

Retruns the SOAPHeader, if set, from the SOAPEnvelope.

Returns: SOAPHeader if there is one else null.

setContent(Source)

public abstract void setContent(javax.xml.transform.Source source)

Set the content of the SOAPEnvelope as a Source

Parameters:

Throws:
IllegalArgumentException - if there is a problem in setting the source.

setSOAPBody(SOAPBody)

public abstract void setSOAPBody(SOAPBody[79] soapBody)
throws SOAPException

Set the SOAPBody part of the envelope.

Parameters:
`SOAPBody` - the soapBody.

Throws:
`SOAPException`[95] - if the SOAPBody being set doesn't confirm to the SOAP specification.

setSOAPHeader(SOAPHeader)

public abstract void setSOAPHeader(`SOAPHeader`[102] soapHeader)
throws SOAPException

Set the SOAPHeader part of the envelope.

Parameters:
`SOAPHeader` - the soapheader.

Throws:
`SOAPException`[95] - if the SOAPHeader being set doesn't confirm to the SOAP specification.

javax.xml.soap
SOAPException

Declaration

public class `SOAPException` extends java.lang.Exception

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--javax.xml.soap.SOAPException
```

All Implemented Interfaces: java.io.Serializable

Direct Known Subclasses: `JAXMException`[43]

Description

An exception that signals that a SOAP exception has occurred. A `SOAPException` object may contain a `String` that gives the reason for the exception, an embedded `Throwable` object, or both. This class provides methods for retrieving reason messages and for retrieving the embedded `Throwable` object.

Typical reasons for throwing a `SOAPException` object are problems such as difficulty setting a header, not being able to send a message, and not being able to get a connection with the provider. Reasons for embedding a `Throwable` object include problems such as input/output errors or a parsing problem e.g. an error in parsing a header.

Member Summary

Constructors

public	`SOAPException()`[96] Constructs a `SOAPException` object with no reason or embedded `Throwable` object.
public	`SOAPException(String)`[96] Constructs a `SOAPException` object with the given `String` as the reason for the exception being thrown.
public	`SOAPException(String, Throwable)`[96] Constructs a `SOAPException` object with the given `String` as the reason for the exception being thrown and the given `Throwable` object as an embedded exception.
public	`SOAPException(Throwable)`[97] Constructs a `SOAPException` object initialized with the given `Throwable` object.

Methods

public Throwable	`getCause()`[97] Returns the `Throwable` object embedded in this `SOAPException` if there is one.
public String	`getMessage()`[97] Returns the detail message for this `SOAPException` object.

Member Summary	
public synchronized Throwable	initCause(Throwable) 97 Initializes the cause field of this SOAPException object with the given Throwable object.

Inherited Member Summary
Methods inherited from class java.lang.Object clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
Methods inherited from class java.lang.Throwable fillInStackTrace, getLocalizedMessage, printStackTrace, printStackTrace, printStackTrace, toString

Constructors

SOAPException()

public SOAPException()

Constructs a SOAPException object with no reason or embedded Throwable object.

SOAPException(String)

public SOAPException(java.lang.String reason)

Constructs a SOAPException object with the given String as the reason for the exception being thrown.

Parameters:

reason - a description of what caused the exception

SOAPException(String, Throwable)

public SOAPException(java.lang.String reason, java.lang.Throwable cause)

Constructs a SOAPException object with the given String as the reason for the exception being thrown and the given Throwable object as an embedded exception.

Parameters:

reason - a description of what caused the exception

cause - a Throwable object that is to be embedded in this SOAPException object

SOAPException(Throwable)

public SOAPException(java.lang.Throwable cause)

Constructs a `SOAPException` object initialized with the given `Throwable` object.

Methods

getCause()

public java.lang.Throwable getCause()

Returns the `Throwable` object embedded in this `SOAPException` if there is one. Otherwise, this method returns `null`.

Returns: the embedded `Throwable` object or `null` if there is none

getMessage()

public java.lang.String getMessage()

Returns the detail message for this `SOAPException` object.

If there is an embedded `Throwable` object, and if the `SOAPException` object has no detail message of its own, this method will return the detail message from the embedded `Throwable` object.

Overrides: java.lang.Throwable.getMessage() in class java.lang.Throwable

Returns: the error or warning message for this `SOAPException` or, if it has none, the message of the embedded `Throwable` object, if there is one

initCause(Throwable)

public synchronized java.lang.Throwable initCause(java.lang.Throwable cause)

Initializes the `cause` field of this `SOAPException` object with the given `Throwable` object.

This method can be called at most once. It is generally called from within the constructor or immediately after the constructor has returned a new `SOAPException` object. If this `SOAPException` object was created with the constructor `SOAPException(Throwable)`$_{97}$ or `SOAPException(String, Throwable)`$_{96}$, meaning that its `cause` field already has a value, this method cannot be called even once.

Parameters:

`cause` - the `Throwable` object that caused this `SOAPException` object to be thrown. The value of this parameter is saved for later retrieval by the `getCause()`$_{97}$ method. A `null` value is permitted and indicates that the cause is nonexistent or unknown.

Returns: a reference to this `SOAPException` instance

Throws:

`IllegalArgumentException` - if `cause` is this `Throwable` object. (A `Throwable` object cannot be its own cause.)

`IllegalStateException` - if this `SOAPException` object was created with `SOAPException(Throwable)`[97] or `SOAPException(String, Throwable)`[96], or this method has already been called on this `SOAPException` object

javax.xml.soap
SOAPFault

Declaration

public abstract class **SOAPFault** extends **SOAPBodyElement**[82]

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement[85]
        |
        +--javax.xml.soap.SOAPBodyElement[82]
              |
              +--javax.xml.soap.SOAPFault
```

Description

SOAPFault carries error and/or status information within a SOAP message. The SOAPFault object is an instance variable in SOAPBody.

Member Summary

Constructors

public	SOAPFault(String, String, String)[100]
public	SOAPFault(String, String, String, String)[100]

Methods

public String	getDetail()[100] Get the fault detail.
public String	getFaultCode()[100] Get the fault code.
public String	getFaultString()[100] Get the fault string.
public void	setDetail(String)[101] Set the fault detail.
public void	setFaultCode(String)[101] Set the fault code.
public void	setFaultString(String)[101] Set the fault string.

Inherited Member Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from class SOAPElement[85]

Inherited Member Summary
addAttribute(String, String, Namespace)[87], addAttribute(String, String, Namespace)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addNamespaceDeclaration(Namespace)[88], createSOAPElement(String, Namespace)[88], getAdditionalNamespaces()[88], getAttributeValue(String)[89], getAttributes()[88], getContent()[89], getEncodingStyle()[89], getName()[89], getNamespacePrefix()[89], getNamespaceURI()[89], getParent()[90], getQualifiedName()[90], setEncodingstyle(String)[90], setParent(SOAPElement)[90]

Constructors

SOAPFault(String, String, String)

public SOAPFault(java.lang.String faultCode, java.lang.String faultString, java.lang.String detail)

SOAPFault(String, String, String, String)

public SOAPFault(java.lang.String faultCode, java.lang.String faultString, java.lang.String detail, java.lang.String actor)

Methods

getDetail()

public java.lang.String getDetail()

Get the fault detail.

getFaultCode()

public java.lang.String getFaultCode()

Get the fault code.

Returns: String the fault code.

getFaultString()

public java.lang.String getFaultString()

Get the fault string.

setDetail(String)

public void setDetail(java.lang.String detail)

Set the fault detail.

setFaultCode(String)

public void setFaultCode(java.lang.String faultCode)

Set the fault code.

Parameters:

`String` - the fault code to be set.

setFaultString(String)

public void setFaultString(java.lang.String faultString)

Set the fault string.

javax.xml.soap
SOAPHeader

Declaration

public abstract class **SOAPHeader** extends **SOAPElement**$_{85}$

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement85
        |
        +--javax.xml.soap.SOAPHeader
```

Description

A SOAP header object represents the contents of the SOAP header element. A SOAP header element consists of XML data that affects the way the application specific content is processed by the message handler. For example, transaction semantics, authentication information etc. can be specified as the content of a SOAP header object. SOAPHeader can only have SOAPHeaderElement as it's immediate children as per the SOAP specification.

See Also: SOAPHeaderElement$_{104}$

Member Summary

Constructors

public	SOAPHeader()$_{103}$

Methods

public abstract void	addContent(SOAPHeaderElement)$_{103}$ Add a SOAPHeaderElement to the SOAPHeader.
public abstract SOAP-HeaderElement	createSOAPHeaderElement(String, Namespace)$_{103}$ Creates a new SOAPHeaderElement

Inherited Member Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from class SOAPElement$_{85}$

Inherited Member Summary
addAttribute(String, String, Namespace)[87], addAttribute(String, String, Namespace)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addNamespaceDeclaration(Namespace)[88], createSOAPElement(String, Namespace)[88], getAdditionalNamespaces()[88], getAttributeValue(String)[89], getAttributes()[88], getContent()[89], getEncodingStyle()[89], getName()[89], getNamespacePrefix()[89], getNamespaceURI()[89], getParent()[90], getQualifiedName()[90], setEncodingstyle(String)[90], setParent(SOAPElement)[90]

Constructors

SOAPHeader()

public SOAPHeader()

Methods

addContent(SOAPHeaderElement)

public abstract void addContent(SOAPHeaderElement[104] element)

Add a SOAPHeaderElement to the SOAPHeader. Note it is only possible to add a SOAPHeaderElement to the SOAPHeader. It is illegal to add a SOAPElement to the SOAPHeader. However only SOAPElement can be appended as children to the SOAPHeaderElement.

Parameters:
SOAPHeaderElement - the SOAPHeaderElement to be appended as a child to the SOAPHeader.

Throws:
IllegalArgumentException - if there is a problem in the header element being added.

createSOAPHeaderElement(String, Namespace)

public abstract SOAPHeaderElement[104] createSOAPHeaderElement(java.lang.String localName, Namespace[77] namespace) throws SOAPException

Creates a new SOAPHeaderElement

Returns: SOAPHeaderElement - the new SOAPHeaderElement created.

Throws:
SOAPException[95]

javax.xml.soap
SOAPHeaderElement

Declaration

public abstract class **SOAPHeaderElement** extends **SOAPElement**$_{85}$

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPElement85
        |
        +--javax.xml.soap.SOAPHeaderElement
```

Description

A SOAPHeaderElement object represents the contents in the SOAPHeader. The immediate children of a SOAP-Header can only be represented as a SOAPHeaderElement. SOAPHeaderElements can consist of other SOAPElements as it's children.

Member Summary

Constructors

public	SOAPHeaderElement()$_{105}$

Methods

public String	getActor()$_{105}$ Returns the uri of the actor.
public boolean	getMustUnderstand()$_{105}$ Returns the mustunderstand attribute associated with the soap header element.
public void	setActor(String)$_{105}$ Set the actor associated with the SOAPHeaderElement.
public void	setMustUnderstand(boolean)$_{106}$ Set the mustunderstand attribute for this particular soap header element.

Inherited Member Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from class SOAPElement$_{85}$

Inherited Member Summary
addAttribute(String, String, Namespace)[87], addAttribute(String, String, Namespace)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addContent(SOAPElement)[87], addNamespaceDeclaration(Namespace)[88], createSOAPElement(String, Namespace)[88], getAdditionalNamespaces()[88], getAttributeValue(String)[89], getAttributes()[88], getContent()[89], getEncodingStyle()[89], getName()[89], getNamespacePrefix()[89], getNamespaceURI()[89], getParent()[90], getQualifiedName()[90], setEncodingstyle(String)[90], setParent(SOAPElement)[90]

Constructors

SOAPHeaderElement()

public SOAPHeaderElement()

Methods

getActor()

public java.lang.String getActor()

Returns the uri of the actor.

getMustUnderstand()

public boolean getMustUnderstand()

Returns the mustunderstand attribute associated with the soap header element.

Returns: boolean the mustunderstand attribute of this header element.

setActor(String)

public void setActor(java.lang.String actorURI)

Set the actor associated with the SOAPHeaderElement.

Parameters:
`String` - actorURI - The uri of the actor.

Throws:
`IlleglalArgumentException` - if there is a problem in setting the actor.

setMustUnderstand(boolean)

public void setMustUnderstand(boolean mustUnderstand)

Set the mustunderstand attribute for this particular soap header element.

Parameters:

`boolean` - mustUnderstand attribute of this header element..

Throws:

`IlleglalArgumentException` - if there is a problem in setting the must understand attribute.

javax.xml.soap
SOAPPart

Declaration

public abstract class **SOAPPart**

```
java.lang.Object
  |
  +--javax.xml.soap.SOAPPart
```

Description

The container for the SOAP specific portion of a `Message` object. The `SOAPPart` object is a MIME part containing the SOAPEnvelope.

A client can access the SOAP Envelope of a `Message` object by calling the method `Message.getSOAPPart`. The following line of code, in which `m` is a `Message` object, retrieves the SOAP part of a message.

```
SOAPPart sp = m.getSOAPPart();
```

`sp.getSOAPEnvelope` can then be used to retrieve the SOAP Envelope itself.

Member Summary

Constructors

public	`SOAPPart()` *108*

Methods

public abstract void	`addMimeHeader(String, String)` *108* Add a MIME header with the specified name and value to this attachment.
public abstract Iterator	`getAllMimeHeaders()` *108* Return all the headers as an Iterator over MimeHeader objects
public String	`getContentId()` *109* Convienience method to get the value of the MIME header Content-Id.
public String	`getContentLocation()` *109* Convienience method to get the value of the MIME header Content-Location.
public abstract Iterator	`getMatchingMimeHeaders(String[])` *109* Return all matching MimeHeader objects
public abstract String	`getMimeHeader(String)` *109* Gets all the values of the header that is identified by the given `String`.
public abstract Iterator	`getNonMatchingMimeHeaders(String[])` *109* Return all non-matching MimeHeader objects
public abstract SOAPEnvelope	`getSOAPEnvelope()` *109* Get the SOAP envelope associated with this `SOAPPart` object.
public abstract void	`removeAllMimeHeadersHeaders()` *110* Remove all the MIME header entries.
public abstract void	`removeMimeHeader(String)` *110* Remove all MIME headers that match the given name.
public void	`setContentId(String)` *110* Convienience method to set the value of the MIME header Content-Id.

Member Summary	
public void	setContentLocation(String) 110 Convienience method to set the value of the MIME header Content-Location.
public abstract void	setMimeHeader(String, String) 110 Change the first header entry that matches name to have value, adding a new header if no existing header matches.

Inherited Member Summary
Methods inherited from class java.lang.Object clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

SOAPPart()

public SOAPPart()

Methods

addMimeHeader(String, String)

public abstract void addMimeHeader(java.lang.String name, java.lang.String value)

Add a MIME header with the specified name and value to this attachment.

Note that RFC822 headers can only contain US-ASCII characters.

Parameters:

`name` - header name

`value` - header value

Throws:

`IllegalArgumentException` - if there was a problem with the specified mime header name and value.

getAllMimeHeaders()

public abstract java.util.Iterator getAllMimeHeaders()

Return all the headers as an Iterator over MimeHeader objects

Returns: MimeHeader objects

getContentId()

public java.lang.String getContentId()

Convienience method to get the value of the MIME header Content-Id.

getContentLocation()

public java.lang.String getContentLocation()

Convienience method to get the value of the MIME header Content-Location.

getMatchingMimeHeaders(String[])

public abstract java.util.Iterator getMatchingMimeHeaders(java.lang.String[] names)

Return all matching MimeHeader objects

Returns: matching MimeHeader objects

getMimeHeader(String)

public abstract java.lang.String[] getMimeHeader(java.lang.String name)

Gets all the values of the header that is identified by the given `String`.

Parameters:
`name` - the name of the header; example: “Content-Type”

Returns: a `String` giving the value for the specified header

See Also: `setMimeHeader(String, String)` 110

getNonMatchingMimeHeaders(String[])

public abstract java.util.Iterator getNonMatchingMimeHeaders(java.lang.String[] names)

Return all non-matching MimeHeader objects

Returns: non-matching MimeHeader objects

getSOAPEnvelope()

public abstract `SOAPEnvelope` 91 getSOAPEnvelope()

Get the SOAP envelope associated with this `SOAPPart` object. Once the SOAP envelope is obtained, the contents of the envelope e.g. SOAPHeader, SOAPFault or SOAPBody can be examined.

removeAllMimeHeadersHeaders()

public abstract void removeAllMimeHeadersHeaders()

Remove all the MIME header entries.

removeMimeHeader(String)

public abstract void removeMimeHeader(java.lang.String header)

Remove all MIME headers that match the given name.

Parameters:

`header` - the string name of the MIME header/s that is to be removed.

setContentId(String)

public void setContentId(java.lang.String contentId)

Convienience method to set the value of the MIME header Content-Id.

Throws:

`IllegallArgumentException` - if there is a problem in setting the content id.

setContentLocation(String)

public void setContentLocation(java.lang.String contentLocation)

Convienience method to set the value of the MIME header Content-Location.

Throws:

`IlleglalArgumentException` - if there is a problem in setting the content location.

setMimeHeader(String, String)

public abstract void setMimeHeader(java.lang.String name, java.lang.String value)

Change the first header entry that matches name to have value, adding a new header if no existing header matches. Remove all matching headers but the first.

Note that RFC822 headers can only contain US-ASCII characters

Parameters:

`name` - header name

`value` - header value

Throws:

`IllegalArgumentException` - if there was a problem with the specified mime header name and value.

CHAPTER JAXM.6

References

- ebXML Transport, Routing & Packaging Specification - Message Service Specification
  - http://www.ebXML.org
- XML Messaging Requirements specification
  - http://www.ietf.org
- MIME-based Secure EDI
  - http://www.ietf.org
- SOAP
  - http://www.w3.org/TR/SOAP
- SOAP Messages with Attachments
  - http://www.w3.org/TR/SOAP-attachments
- Java Activation Framework Version 1.0a
  - http://www.javasoft.com
- Java API for XML Processing - Version 1.1 Final Release
  - http://www.javasoft.com