SeeBeyond™ eBusiness Integration Suite

e*Way Intelligent Adapter for SOAP User's Guide

Release 4.5.2

Version 20011231141718.

Contents

Chapter 1

Introduction

This chapter introduces you to SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way™ Intelligent Adapter for SOAP (SOAP e*Way). It also provides an overview of the Simple Object Access Protocol (SOAP) and how to use this e*Way.

1.1 SOAP e*Way: Overview

The SOAP e*Way enables the e*Gate system to exchange data with Internet and Web Services applications that exchange information using SOAP. This e*Way is enabled by the Java programming language.

1.1.1 Introduction to SOAP

SOAP, based on the Extensible Markup Language (XML), is a lightweight protocol for the exchange of information in a distributed, decentralized environment. SOAP specifies how to create an XML file and the encoding for HTTP. The protocol enables an application to communicate over the Internet regardless of the operating system (OS), object model, or implementation language.

SOAP is similar to IIOP, CORBA, and RMI. However, in contrast to these protocols, SOAP has been designed to be fire-wall friendly, lightweight, and easy to implement.

Conventions and Specifications

SOAP defines a set of conventions for the following purposes:

- To format its own messages
- To contain rules for carrying a SOAP message within or on top of another protocol
- To process SOAP messages along the SOAP message path

SOAP specifications can be found on World Wide Web Consortium (W3C) Web site as follows:

- SOAP Version 1.2, Part 1: Messaging Framework:

  http:\\www.w3.org\TR\2001\WD-soap12-part1-20011002

- SOAP Version 1.2, Part 2: Adjuncts:

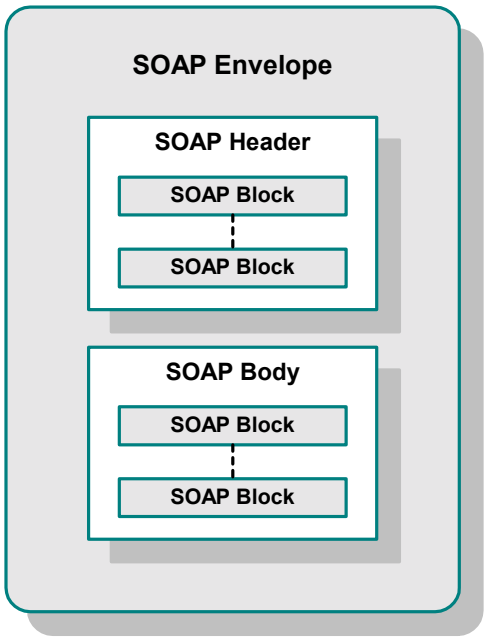
  http:\\www.w3.org\TR\soap12-part2

Note: *The SOAP e*Way is compatible with both SOAP versions 1.1 and 1.2.*

SOAP Messaging

Figure 1 shows a diagram of the SOAP message components.

Figure 1 Soap Message Components



SOAP messages (see Figure 1) consist of the following major parts:

- A required SOAP envelope that marks the start and end of the SOAP message and defines a framework for describing what is in a message and how to process it
- An optional SOAP header that carries general information about the SOAP message in one or more header blocks
- A required SOAP body made up of one or more blocks that carry the actual message payload

SOAP messages also specify:

- A set of encoding rules for expressing instances of application-defined data types
- A convention for representing remote procedure calls and responses

Additionally, a special type of SOAP body block, a SOAP fault, is used to carry error and/or status information. If it is present, a SOAP fault (body block) occurs only once.

Example: SOAP Message

Sample request envelope:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

```
<SOAP-ENV:Body>
<ns1:getQuote xmlns:ns1="urn:xmethods-delayed-quotes" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<symbol xsi:type="xsd:string">IBM</symbol>
</ns1:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample response envelope:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:getQuoteResponse xmlns:ns1="urn:xmethods-delayed-quotes" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:float">133.625</return>
</ns1:getQuoteResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A universal SOAP standard is in "working draft" status with W3C, but it is backed by many leading organizations. SOAP is a major building block for emerging Web Services. Web Services are the next generation model for businesses using the Internet to allow business services and functions to be accessed by other applications across the Internet.

You can learn more about SOAP by visiting the following Web sites:

- **http:\\www.xmethods.net**
- **http:\\www.webservices.org**
- **http:\\www.ibm.com\developerworks\webservices**
- **http:\\msdn.microsoft.com\soap**
- **http:\\www.develop.com\soap**
- **http:\\www.soapware.org**

1.1.2 e*Way Components and Features

This section provides an overview of the SOAP e*Way including its basic components and features.

Basic Components

The SOAP e*Way includes the following components:

- **stcsoap.jar** contains the logic required to implement SOAP e*Way functions.
- **stceway.exe** provides all the basic e*Way functions and invokes **stcsoap.jar**.
- Third-party libraries, including **xerces.jar**, **activation.jar**, **mail.jar**, **jsafe.jar**, **certj.jar**, and **xalan.jar**, are listed in **Table 1 on page 16**.

Supported Features

This SOAP e*Way version provides SOAP receiver and sender synchronous (RPC) and asynchronous messaging support for the messaging framework, using HTTP transport bindings.

In addition, this version of the SOAP e*Way supports the following features:

- SOAP version 1.2 and 1.1 messaging
- Message transport on top of HTTP(S)
- Messages with attachments
- Digital signatures

Note: *When referring specifically to HTTP clear, this guide uses the term HTTP. For HTTP over SSL, that is, secure HTTP, it uses the term HTTPS. For generic HTTP that can be either clear or secure, it uses the term HTTP(S).*

See **Chapter 7** for more information on the SOAP e*Way's additional features.

1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have high-level knowledge of the Java Programming Language; to have high-level knowledge of Windows and UNIX operations and administration; to be thoroughly familiar with SOAP protocol and to be thoroughly familiar with Windows-style GUI operations.

1.3 Supported Operating Systems

The SOAP e*Way is available on the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3
- HP-UX 11.0 and HP-UX 11i
- Compaq Tru64 V4.0F and V5.0A

1.4 System Requirements

To use the SOAP e*Way, you need:

- An e*Gate Participating Host, version 4.5.1 or later
- A TCP/IP network connection
- Pentium-class 866 MHz CPU

1.5 External System Requirements

The SOAP e*Way supports the following external systems:

- To use the e*Way, you need a SOAP service on the network/Internet
- To use the SOAP e*Way sample schemas, you need access to the following Web site:

  www.xmethods.net

Additional Requirements

Use of the SOAP e*Way requires using the e*Gate API Kit and the HTTP(S) e*Way Intelligent Adapter. See the *e*Gate API Kit User's Guide* and the *HTTP(S) e*Way Intelligent Adapter User's Guide* for more information.

Chapter 2

Installation

This chapter explains how to install the e*Way Intelligent Adapter for SOAP.

2.1 Windows NT or Windows 2000

2.1.1 Pre-installation

- Exit all Windows programs before running the setup program, including any anti-virus applications.
- You must have Administrator privileges to install this e*Way.

2.1.2 e*Way Installation Procedure

To install the SOAP e*Way on a Windows NT or Windows 2000 system

1. Log in as an Administrator on the workstation where you want to install the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's **Auto-run** feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the **setup.exe** file on the CD-ROM drive.
4. After the **InstallShield** setup application launches, follow the on-screen instructions to install the e*Way.

Note: *When you select the SOAP e*Way, the installation automatically selects the HTTP(S) e*Way and the e*Gate Integrator API Kit for installation.*

Be sure to install the e*Way files in the suggested **\client** installation directory. The installation utility detects and suggests the appropriate installation directory.

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.

2.2 UNIX

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under your name with which you wish to own the e*Way files. Be sure that this user has sufficient privileges to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install the SOAP e*Way on a UNIX system

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type:

   cd /cdrom/setup
4. Start the installation script by typing:

   setup.sh
5. A menu of options appears. Select the **e*Gate Add-on Applications** option. Then, follow any additional on-screen directions.

Note: *When you select the SOAP e*Way, the installation automatically selects the HTTP(S) e*Way and the e*Gate Integrator API Kit for installation.*

Be sure to install the e*Way files in the suggested **\client** installation directory. The installation utility detects and suggests the appropriate installation directory.

Caution: *Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.*

Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.

2.3 After Installation

The SOAP e*Way installation automatically installs the e*Gate API Kit. After installing the e*Way, you must follow the instructions in the *e*Gate API Kit User's Guide* and copy the appropriate files to their Web server directories before you can use the SeeBeyond JMS IQ Service.

Note: *See the* ***e*Gate API Kit User's Guide*** *for more information on this feature.*

In addition, to run some of the sample e*Gate schemas, you must place the appropriate ASP files included in the sample schema .**zip** file in the correct directories. The configuration of your Web server determines these directory locations.

2.4 Files/Directories Created by the Installation

Whether for Windows or UNIX, the SOAP e*Way installation installs the files shown in Table 1 within the e*Gate directory tree. Files are installed within the **egate\client** tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Installation Files and Directories

e*Gate Directory	File
client\classes\	stcsoap.jar
client\Thirdparty\jaf-1.0.1\classes\	activation.jar
client\Thirdparty\javamail-1.2\classes\	mail.jar
server\registry\repository\default\classes\	stcsoap.jar stcutil.jar
server\registry\repository\default\configs\ewsoap\	ewsoap.def
server\registry\repository\default\Thirdparty\jaf-1.0.1\classes\	activation.jar
server\registry\repository\default\Thirdparty\javamail-1.2\classes\	mail.jar
client\Thirdparty\RSA\certJ_2.0.1\classes\	certj.jar xalan.jar
client\Thirdparty\RSA\cryptoj_3.3\classes\	jsafe.jar
server\Thirdparty\server\repository\default\Thirdparty\RSA\certJ.2.0.1\classes	certj.jar xalan.jar
server\Thirdparty\server\repository\default\Thirdparty\RSA\cryptoj_3.3\classes	jsafe.jar
eGate\client\etd\ewsoap	SOAPSimple.xsc SOAPSimple.jar

Table 1 Installation Files and Directories (Continued)

e*Gate Directory	File
eGate\client\etd	soap.ctl soapwizard.ctl stcewsoap.ctl
server\registry\repository\default\	addonconnpt.ini

Chapter 3

Multi-Mode e*Way Configuration

This chapter describes how to configure the e*Gate Integrator's Multi-Mode e*Way Intelligent Adapter.

3.1 Multi-Mode e*Way Properties

Set the Multi-Mode e*Way properties using the e*Gate Enterprise Manager graphical user interface (GUI).

To set properties for a new Multi-Mode e*Way

1. Select the Navigator pane's Components tab in the Main window of the Enterprise Manager.
2. Open the host and Control Broker where you want to create the e*Way.
3. On the Palette, click on the icon to create a new e*Way.
4. Enter the name of the new e*Way, then click **OK**.
5. Select the new component, then click the Properties icon to edit its properties.

   The e*Way Properties dialog box opens
6. Click **Find** beneath the **Executable File** field, and select an executable file (**stceway.exe** is located in the **\bin** directory).
7. Under the **Configuration File** field, click **New**.

   The e*Way Editor GUI opens.
8. When the **Settings** page opens, set the configuration parameters for this e*Way's configuration file.
9. After selecting the desired parameters, click **Save** on the **File** menu to save and close configuration (**.cfg**) file. This action also closes the e*Way Editor.
10. Click **OK** to close the e*Way Properties dialog box and save the properties.

After setting properties for the Multi-Mode e*Way, you must set the component's configuration parameters. These parameters are explained under the following section:

- JVM Settings

3.2 JVM Settings

To correctly configure the Multi-Mode e*Way, you must configure the Java Virtual Machine (JVM) settings. This section explains the configuration parameters in the e*Way Editor GUI, which control these settings.

JNI DLL Absolute Pathname

Description

Specifies the absolute path name to where the JNI **.dll** (Windows) or shared library (UNIX) file is installed by the *Java 2 SDK 1.3*, on the Participating Host. This parameter is *mandatory*.

Required Values

A valid path name.

Additional Information

The JNI **.dll** or shared library file name varies, depending on the current operating system (OS). The following table lists the file name by OS:

Operating System	Java 2 JNI .dll or Shared Library Name
Windows NT/2000	jvm.dll
Solaris	libjvm.so
HP-UX	libjvm.sl
AIX	libjvm.a
Compaq	libjvm.so

The value assigned can contain a reference to an environment variable, by enclosing the variable name within a pair of "%" symbols, for example:

```
%MY_JNIDLL%
```

Such variables can be used when multiple Participating Hosts are used on different OS/platforms.

Caution: *To ensure that the JNI **.dll** file loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory, which contain shared library or **.dll** files.*

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the JVM.

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

If left unset, no paths are prepended to the CLASSPATH environment variable.

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the JVM. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) will be set.

Note: *All necessary JAR and ZIP files needed by both e*Gate and the JVM must be included. It is advised that the* ***CLASSPATH Prepend*** *parameter should be used.*

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

Specifies whether the path is appended for the CLASSPATH environmental variable to jar and zip files needed by the JVM.

Required Values

YES or **NO**. The configured default is YES.

Initial Heap Size

Description

Specifies the value for the initial heap size in bytes. If set to 0 (zero), the preferred value for the initial heap size of the JVM will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Heap Size

Description

Specifies the value of the maximum heap size in bytes. If set to 0 (zero), the preferred value for the maximum heap size of the JVM will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for Native Threads

Description

Specifies the value of the maximum stack size in bytes for native threads. If set to 0 (zero), the default value will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for JVM Threads

Description

Specifies the value of the maximum stack size in bytes for JVM threads. If set to 0 (zero), the preferred value for the maximum heap size of the JVM will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Class Garbage Collection

Description

Specifies whether the Class Garbage Collection will be done automatically by the JVM. The selection affects performance issues.

Required Values

YES or **NO**.

Garbage Collection Activity Reporting

Description

Specifies whether garbage collection activity will be reported for debugging purposes.

Required Values

YES or **NO**.

Asynchronous Garbage Collection

Description

Specifies whether asynchronous garbage collection activity will be reported for debugging purposes.

Required Values

YES or **NO**.

Report JVM Info and all Class Loads

Description

Specifies whether the JVM information and all class loads will be reported for debugging purposes.

Required Values

YES or **NO**.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler is disabled.

Required Values

YES or **NO**.

Remote debugging port number

Description

Specifies whether to allow remote debugging of the JVM.

Required Values

YES or **NO**.

Suspend Option for Debugging

Description

Indicates whether to suspend Option for Debugging on JVM startup.

Required Values

YES or **NO**.

Chapter 4

e*Way Connection Configuration

This chapter explains how to configure e*Way Connections for the e*Way Intelligent Adapter for SOAP.

4.1 Configuring e*Way Connections

Set up e*Way Connections using the e*Gate Enterprise Manager graphical user interface (GUI).

To create and configure e*Way Connections

1. In the Enterprise Manager's **Navigation** pane**,** select the **Component** tab.
2. Select the **e*Way Connections** folder.
3. On the palette, click on the icon to create a new **e*Way Connection**.

   The **New e*Way Connection Component** dialog box appears.
4. Enter a name for the **e*Way Connection**, then click **OK**. For the examples given in **Chapter 5**, the name is **SoapConnection.**

   An icon for your new e*Way Connection appears in the Navigation pane.
5. Double-click on the new **e*Way Connection** icon.

   The **e*Way Connection Properties** dialog box appears.
6. From the **e*Way Connection Type** drop-down box, select (for the examples) **SOAP**.
7. Enter **-1** for the **Event Type "get" interval** in the dialog box provided.
8. From the **e*Way Connection Configuration File**, click **New** to open the e*Way Editor GUI.

Note: *To use an existing file, click* ***Find****.*

9. Use the e*Way Editor to create a new configuration file for this e*Way Connection. Do this operation by selecting the appropriate configuration parameters available in the GUI.
10. When you are finished, close the e*Way Editor and save the new configuration file. For the examples given in **Chapter 5**, the file name is **SoapConnection.cfg.**

The rest of this chapter explains the SOAP e*Way Connection configuration parameters as follows:

4.2 Configuration Parameters

This section explains the configuration parameters for the SOAP e*Way Connection.

4.2.1 Connector

The parameters in the **Connector** section allow the Collaboration engine to identify the e*Way Connection.

Type

Description

Specifies the type of e*Way Connection.

Required Values

SOAP. The value defaults to **SOAP**.

Class

Description

Specifies the class name of the SOAP connector object.

Required Values

A valid package name. The default is **com.stc.eways.SOAP.SOAPConnector**.

Property.Tag

Description

Identifies the data source. This parameter is required by the current **EBobConnectorFactory**.

Required Values

A valid data source package name.

4.2.2 Transport Binding

Description

The parameters in the **Transport Binding** section configure the transport binding used by the SOAP e*Way when sending messages to the SOAP server.

Transport Type

Description

A transport binding to be used for posting SOAP messages.

Required Values

HTTP or **HTTPS**. The value defaults to **HTTP**.

SOAPAction URI

Description

This parameter specifies the **SOAPAction** URI header and is used only if the transport type is HTTP or HTTP(S).

Required Values

The value defaults to **com.stc.eways.soap.SOAP**, which is the only option.

SOAP Style

Description

This parameter specifies the SOAP style to use when interacting with a SOAP server.

Required Values

You can select either **RPC** or **Document** style. With **RPC** style, you can expect to receive a valid SOAP message or a valid MIME message (if the SOAP message has attachments). The valid message is unmarshaled into the **SOAPResponse** node of the SOAP ETD. With **Document** style, no response is expected. Calling **marshal** on the **SOAPResponse** node results in an empty SOAP document.

By default, **SOAP Style** is set to **RPC**. This value can be overridden by methods used in the SOAP ETD.

4.2.3 Security

Description

The parameters in this **Security** section allow you to specify the keys and certificates used by the SOAP e*Way to sign and verify SOAP messages.

KeyStore

Description

This parameter sets the default KeyStore file for use by the KeyManager. If the default KeyStore is not specified with this method, the KeyStore managed by KeyManager is empty.

Required Values

A valid KeyStore file name.

KeyStore Type

Description

This parameter sets the default KeyStore type. If the default KeyStore type is not set here, the default KeyStore type **JKS** is used. Other possible types include, for example, **PKCS12**.

Required Values

The name of a valid KeyStore type.

KeyStore Password

Description

This parameter sets the default KeyStore password. If the default KeyStore password is not set here, then the default KeyStore password is assumed to be " ".

Required Values

A valid KeyStore password.

Default Alias

Description

This parameter sets the alias name for the private key and the digital certificate. All entries in a KeyStore are identified by an alias. This parameter identifies the location of the private key and the digital certificate in the KeyStore. If **Default Alias** is not set, the default is assumed to be " ".

Required Values

A valid alias name for the private key and the digital certificate.

Signature Algorithm

Description

This parameter sets the signature algorithm to use when signing SOAP documents. One of these two algorithms *must* be set for the authentication to work. The default algorithm is **dsa-sha1**.

Required Values

The appropriate algorithm, either **dsa-sha1** or **rsa-sha1**.

4.2.4 Transport Level Retry

Description

The parameters in the **Transport Level Retry** section are related to the retry of transport posting. These parameters are used by the **SendToSOAPServer** function when it encounters errors at the transport level.

Timeout in Seconds

Description

This parameter is reserved for future use. Currently, the SOAP e*Way relies on the HTTP server to which the e*Way is posting to for time-out functionality.

Required Values

The number of seconds considered appropriate before timing out.

Retry Condition

Description

This parameter specifies the condition under which a retry of the transport posting is to be carried out. If **On Timeout Only** is chosen, the posting is retried only if the failure is due to a timeout on the connection. If **On Any Transport Failure** is chosen, the posting is retried on any transport failure.

Required Values

On Timeout Only or **On Any Transport Failure**. The default is **On Timeout Only**.

Number of Seconds to Wait Before Retry

Description

This parameter specifies the number of seconds to wait before the next retry of the transport posting. The e*Way will sleep through this period of time.

Required Values

The number of seconds considered appropriate before retrying the transport posting.

Maximum Retries

Description

This parameter specifies the maximum number of transport level retries the e*Way carries out before giving up and returning the appropriate status.

Required Values

The number of retries considered appropriate before giving up.

4.2.5 HTTP

This **HTTP** section contains a set of top-level parameters used by HTTP.

DefaultUrl

Description

Specifies the default URL to be used. If HTTPS protocol is specified, Secured Sockets Layer (SSL) must be configured. See the **"Using Secured Sockets Layer" on page 129**.

Required Values

A valid URL.

Additional Information

You must include the full URL, for example:

http://www.seebeyond.com

or

http://google.yahoo.com/bin/query

AllowCookies

Description

Specifies whether cookies sent from servers are stored and sent on subsequent requests. If cookies are not allowed, sessions are not supported.

Required Values

Yes or **No**.

ContentType

Description

Specifies the request content type.

Required Values

A string; the default is:

application/x-www-form-urlencoded

If you are sending other forms of data, set this parameter to the appropriate content type, for example:

text/xml

AcceptType

Description

Specifies the parameters for the **AcceptType** request header.

Required Values

A string. For example **text/html, text/plain, text/xml**, and so on.

4.2.6 Proxies

The **Proxies** parameters in this section specify the information required for the e*Way Connection to access the external systems through a proxy server.

Note: *When using proxy servers with Internet Information Services (IIS) Web servers, you must configure the proxy and IIS servers to release connections in a timely manner. Some proxies use **Keep-Alive** HTTP headers to keep connections open. If you cannot configure the proxy and IIS servers to release connections quickly, do not configure the IIS server with **Keep-Alive** headers. The SOAP e*Way does not use **Keep-Alive** headers and is therefore unaware when the proxy is keeping the connection alive.*

UseProxy

Description

Specifies whether an HTTP or HTTPS proxy is being used. If you set this parameter to HTTP, an HTTP proxy for a non-secured connection is used. If HTTPS is selected, an HTTPS proxy for a secured connection is used. Select **No** if a proxy is not used. See the following configuration parameters: **HttpProxyHost, HttpProxyPort, HttpsProxyHost, HttpsProxyPort, UserName,** and **PassWord** in this section.

Required Values

HTTP, **HTTPS**, or **No**.

HttpProxyHost

Description

Specifies the HTTP proxy host name to which to delegate requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy host for non-secured HTTP connections. To turn on proxy use, see the **UseProxy** configuration parameter.

Required Values

A valid HTTP proxy host name.

HttpProxyPort

Description

Specifies the HTTP proxy port to which requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This parameter sets the proxy port for non-secured HTTP connections. To turn on proxy use, see the **UseProxy** configuration parameter.

Required Values

A valid HTTP proxy port number.

HttpsProxyHost

Description

Specifies the HTTPS proxy host to which requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy port for secured HTTP connections. To turn on proxy use, see the **UseProxy** configuration parameter.

Required Values

A valid HTTPS proxy host number.

HttpsProxyPort

Description

Specifies the HTTPS proxy port to which requests to an HTTP server or reception of data from an HTTP server may be delegated to a proxy. This sets the proxy port for secured HTTP connections. To turn on proxy use, see the **UseProxy** configuration parameter.

Required Values

A valid HTTPS proxy port name.

UserName

Description

Specifies the user name necessary for authentication to access the proxy server. To turn on proxy use, see the **UseProxy** configuration parameter.

Required Values

A valid user name.

Additional Information

The user name is required by URLs that require **HTTP Basic Authentication** to access the site.

Important: *Enter a value for this parameter **before** you enter a value for the **PassWord** parameter.*

PassWord

Description

Specifies the password corresponding to the user name specified previously.

Required Values

The appropriate password.

Important: *Be sure to enter a value for the **UserName** parameter before entering the **PassWord** value.*

4.2.7 HttpAuthentication

The **HttpAuthentication** parameters in this section are used to perform HTTP authentication.

UseHttpAuthentication

Description

Specifies whether standard HTTP authentication is used. This is used when the Web site requires user name and password authentication. If this parameter is selected, the **UserName** and **PassWord** configuration parameters must be set. See **UserName** and **PassWord** configuration parameters in this section.

Required Values

Yes or **No**.

UserName

Description

Specifies the user name for standard HTTP authentication. See the **UseHttpAuthentication** configuration parameter.

Required Values

A valid user name.

Important: *Enter a value for this parameter **before** you enter a value for the **PassWord** parameter.*

PassWord

Description

Specifies the password associated with the specified user name for standard HTTP authentication. See **UseHttpAuthentication** configuration parameter.

Required Values

A valid password.

Important: *Be sure to enter a value for the* ***UserName*** *parameter before entering the* ***PassWord****.*

4.2.8 SSL

The parameters in this section control the information required to set up an SSL connection via HTTP.

UseSSL

Description

Specifies whether SSL needs to be configured in order to use the HTTPS protocol. If this parameter is set to **Yes**, at least **HttpsProtocolImpl** and **Provider** must be given.

Required Values

Yes or **No**.

HttpsProtocolImpl

Description

Specifies the package that contains the HTTP(S) protocol implementation. This specification adds the HTTP(S) **URLStreamHandler** implementation by including the handler's implementation package name to the list of packages searched by the Java URL class. The default value specified is the package that contains the Sun Microsystems reference implementation of the HTTPS **URLStreamHandler**.

Required Values

A valid package name. The default is **com.sun.net.ssl.internal.www.protocol**. This parameter is mandatory if you are using HTTP(S).

Provider

Description

Specifies the Cryptographic Service Provider. This will add a JSSE provider implementation to the list of provider implementations. The default value specified is the Sun Microsystems reference implementation of the Cryptographic Service Provider **SunJSSE**.

Required Values

A valid provider name. The default is **com.sun.net.ssl.internal.ssl.Provider**. This parameter is mandatory if you are using HTTP(S).

X509CertificateImpl

Description

Specifies the implementation class of the **X509Certificate**.

Required Values

A valid package location. For example, if the implementation class is called, **MyX509CertificateImpl**, and it resides in the **com.radcrypto** package, specify **com.radcrypto.MyX509CertificateImpl**.

SSLSocketFactoryImpl

Description

Specifies the implementation class of the SSL socket factory.

Required Values

A valid package location. For example, if the implementation class is called **MySSLSocketFactoryImpl** and it resides in the **com.radcrypto** package, specify **com.radcrypto.MySSLSocketFactoryImpl**.

SSLServerSocketFactoryImpl

Description

Specifies the implementation class of the SSL server socket factory.

Required Values

A valid package location. For example, if the implementation class is called **MySSLServerSocketFactoryImpl** and it resides in **com.radcrypto** package, specify **com.radcrypto.MySSLServerSocketFactoryImpl**.

KeyStore

Description

Specifies the default KeyStore file for use by the KeyManager. If the default KeyStore is not specified with this method, the KeyStore managed by KeyManager is empty.

Required Values

A valid package location.

KeyStoreType

Description

Specifies the default KeyStore type. If the default KeyStore type is not set by this method, the default KeyStore type, **JKS** is used.

KeyStorePassword

Description

Specifies the default KeyStore password. If the default KeyStore password is not set by this method, the default KeyStore password is assumed to be " ".

TrustStore

Description

Specifies the default TrustStore. If the default TrustStore is not set here, then a default TrustStore search is performed. If a TrustStore named **<java-home>/lib/security/jssecacerts** is found, it is used. If not, a search for a TrustStore name **<java-home>/lib/security/cacerts** is made, and used if located. If a TrustStore is not found, the TrustStore managed by the TrustManager is a new empty TrustStore.

Required Values

A valid TrustStore name.

TrustStoreType

Description

Specifies the default TrustStore type.

Required Values

A valid TrustStore type.

TrustStorePassword

Description

Specifies the default TrustStore password. If the default TrustStore password is not set by this method, the default TrustStore password is " ".

KeyManagerAlgorithm

Description

Specifies the default KeyManager algorithm name to use. For example, the default KeyManager algorithm used in the Sun Microsystems reference implementation of JSSE is **SunX509**.

Required Values

A valid KeyManager algorithm name.

TrustManagerAlgorithm

Description

Specifies the default TrustManager algorithm name to use. For example, the default TrustManager algorithm used in the Sun Microsystems reference implementation of JSSE is **SunX509**.

Required Values

A valid TrustManager algorithm name.

4.2.9 Server Information

The parameters in this section allow you to enter server information for the e*Way.

server name

Description

Specifies the host name of the e-mail server.

Required Values

A valid server host name.

port number

Description

Specifies the port number of the database server.

Required Values

A valid database host name.

user name

Description

Specifies the user's log-in name.

Required Values

A valid user log-in name.

password

Description

Specifies the user's log-in password.

Required Values

A valid user log-in password.

Chapter 5

Implementation

This chapter explains how to implement sample schemas for the e*Way Intelligent Adapter for SOAP, both the SOAP sender and receiver.

5.1 SOAP e*Way: Architecture Overview

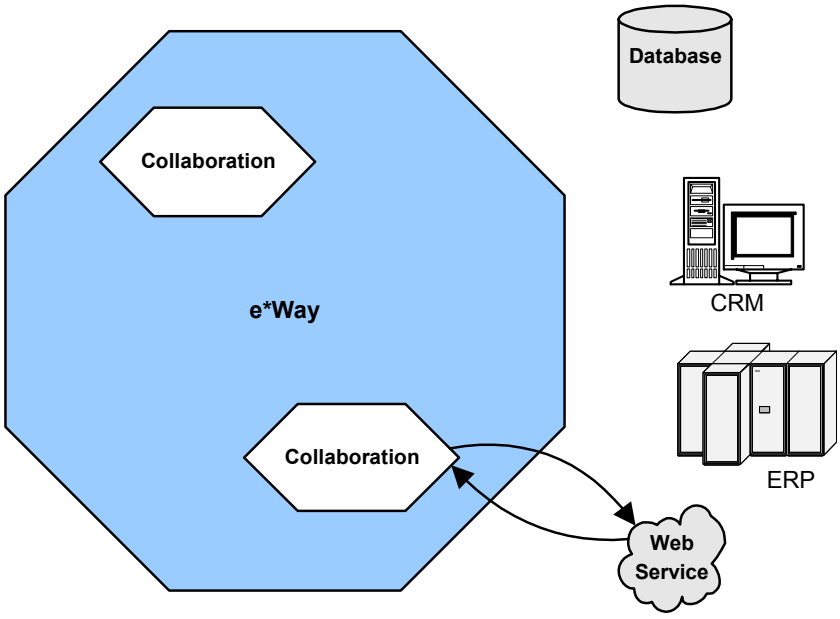
SOAP messaging is essentially the delivery of a message from a SOAP sender to the ultimate SOAP receiver. SOAP messages can be asynchronous, or they can be combined to form a SOAP request/response synchronous message exchange.

This section describes the architectural framework for the implementation of SOAP receivers and senders, regardless of the asynchronous or synchronous nature of the message exchange.

5.1.1 SOAP Sender

Figure 2 shows a diagram of the basic SOAP sender setup in e*Gate.

Figure 2 SOAP Sender e*Gate Setup



An e*Gate Collaboration that sends a SOAP message is implemented using an e*Gate Multi-Mode e*Way. The SOAP e*Way functionality is used by the Multi-Mode e*Way, along with a SOAP e*Way Connection.

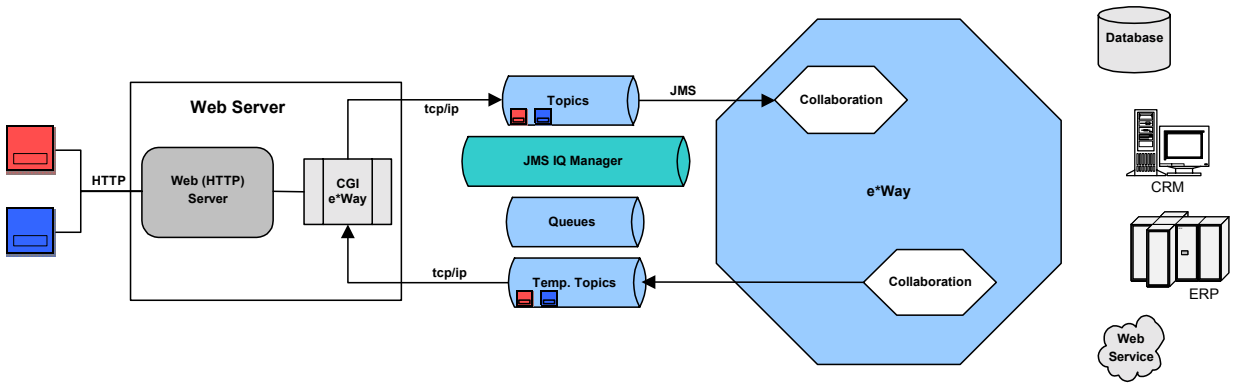
You can specify the SOAP endpoint, or URL, in the e*Way Connection or dynamically in the Collaboration Rules. Additional HTTP binding and SOAP configuration parameters are specified in the e*Way Connection configuration. The SOAP e*Way supports sending request/reply messages, as well as "fire-and-forget" asynchronous messages.

The installation of the SOAP e*Way includes all of the necessary files for sending SOAP messages. Included are the SOAP e*Way and wizard, the HTTP/S e*Way, and third-party libraries.

5.1.2 SOAP Receiver

Figure 3 shows a diagram of the basic SOAP receiver setup in e*Gate.

Figure 3 SOAP Receiver e*Gate Setup



In addition to the SOAP e*Way, implementing an e*Gate Collaboration that receives a SOAP message requires:

- Use of a Web server, such as Apache/Tomcat, iPlanet, or MS IIS
- Either a Web-server e*Way such as the CGI e*Way or the e*Gate API Kit

The e*Gate API Kit provides a JMS application programming interface (API) to the SeeBeyond JMS IQ Manager. In addition, the Web server (or application server) provides the HTTP daemon "listener" facility. You can use either a Web-server e*Way or the JMS IQ Manager (used in this chapter's example) to pass SOAP messages into the e*Gate system.

Figure 3 on page 37 illustrates the components of a SOAP request/response message where an e*Gate Collaboration is providing the SOAP service. External clients request the SOAP service using an end-point URL within the Web server space. Apache/Tomcat is an easily available Web server and is commonly used. Apache/Tomcat is included with some SeeBeyond products, but the SOAP Web server can be any HTTP server.

An e*Gate user implementing the SOAP service can use either:

- A plug-in provided with the SOAP e*Way installation, for example, a JavaServer Page (JSP) that implements the e*Gate API Kit for JMS (API); see the example in this chapter
- User-created code as a base you can use to build custom applications

Web Server Logical Steps

The logical steps on the Web-server side are:

- Capture the HTTP data.
- Create a JMS temporary topic.
- Populate the **JMSReplyTo** header field with the temporary topic.
- Publish the HTTP data to a known topic or queue.
- Subscribe to the temporary topic.
- Wait for the reply message.
- Return the message as the HTTP response body.

Note: *JMS temporary topics and the **JMSReplyTo** header field are JMS features used in a request/reply solution. A temporary topic is a unique, dynamically created topic that is only active for the duration of the connection and is guaranteed to be unique across all connections. Temporary topics are associated only with the message server that the client is in session with. Any client can publish messages to a temporary topic, but only the client connection that created the temporary topic can subscribe to it.*

An e*Gate Collaboration subscribes to the known topic or queue, publishes to the temporary topic found in the **JMSReplyTo** header field, another IQ, or another Collaboration. Depending on the complexity of the service implementation, one or more Collaborations can be involved in processing a SOAP request.

A Collaboration uses a SOAP Event Type Definition (ETD) and the SOAP e*Way to unmarshal the received SOAP message. The Collaboration Rules for the Collaboration use the SOAP ETD to create the response SOAP message. This SOAP message is published to the JMS temporary topic found in the **JMSReplyTo** header field.

e*Gate System Logical Steps

The logical steps on the e*Gate side are:

- Create a schema.
- Create Event Types.
- Create the ETD that receives the message from the JMS IQ Manager.
- Create the SOAP ETD to be used for processing the request and response.
- Create the ETD that sends the reply to the temporary JMS topic.
- Create Collaboration Rules as follows:
  - Drag the source JMS payload onto the SOAP request.
  - Define Java rules to create the SOAP response.
  - Drag the SOAP response onto the JMS **ReplyTo** topic.
- Create e*Way Connections.

This section has described implementing the JMS interface as a request/reply schema. Request/reply is used for both synchronous SOAP request/response messages, as well as asynchronous SOAP messages that expect a SOAP response status message.

To receive true asynchronous "fire-and-forget" SOAP messages, you can implement the JMS publish/subscribe (pub/sub) schema. For more information on the e*Gate API Kit for the JMS (API), see the *e*Gate API Kit Developer's Guide*.

SOAP Services

The implementation of the SOAP e*Way begins with the selection of SOAP service. For the purposes of this chapter, a publicly available SOAP service is used to illustrate the configuration steps for implementing the SOAP e*Way in SOAP RPC style.

An RPC-style SOAP message is a synchronous request/reply process where the SOAP e*Way executes a remote SOAP service by passing input parameters and waiting for output parameters. For a service list of the many publicly available SOAP services, see the following Web site:

www.xmethods.net

5.2 SOAP Sender Implementation

This section explains how to implement a sample schema for the SOAP e*Way, for the SOAP sender.

5.2.1 SOAP Sender Schema: Overview

This section illustrates the SOAP service interface to Altavista's BabelFish service. The service entry is located at:

http:\\www.xmethods.net\detail.html?id=14

Schema Operation

The sample BabelFish schema does the following operations:

- Retrieves text from a text file on any platform
- Transforms the text into a SOAP message
- Posts the SOAP message to a SOAP server that translates the text of the message into a different language (by default, the SOAP server translates an English message into French); for this sample, the SOAP server is the **www.xmethods.net** Web site
- Receives the translated text from the SOAP server
- Publishes the translated text to a different file

Schema Input Data

The following text is the input data used for this sample schema:

```
Good morning
```

Schema Output Data

The SOAP BabelFish service passes as request, or input parameters, a value for **translationmode** and **sourcedata**. The response, or output parameter, is the value for the return data, for example:

translationmode : en_fr

sourcedata : good morning

return : bonjour

Schema Components

This sample BabelFish schema implementation consists of the following components:

e*Ways

- **Feeder** receives text from an external source, applies the e*Gate Pass Through Collaboration Service, and publishes the information to an Intelligent Queue (IQ) that stores inbound data.
- **SOAPBabelFishClient** applies extended Java Collaboration Rules to an inbound Event to perform the desired business logic. In this case, the e*Way translates the inbound Event into a SOAP message, posts the SOAP message to a SOAP server, receives a translated text response from the SOAP server, and publishes the response to an IQ.
- **Eater** receives the outbound message from an IQ and publishes it (via Pass Through again) to a file.

Event Types

- **Feeder_In_Event** contains raw data from the input file.
- **Feeder_Out_Event** contains raw data from the input file.
- **SOAP_BabelFish_Event** contains the request data, the response data (if any), and the methods used to manipulate the data.
- **Eater_In_Event** contains the translated data.
- **Eater_Out_Event** contains the translated data.

Collaboration Rules

- **FeederCollaboration** is associated with the **Feeder** e*Way and is used for receiving the input Event.
- **SOAPBabelFishClient** is associated with the **SOAPBabelFishClient** e*Way and is used to perform the transformation process, send the Event to the SOAP server, and receive a response from the SOAP server.
- **EaterCollaboration** is associated with the **Eater** e*Way and is used for sending the Event to the output file.

IQs

- **In_Q** receives data from the **Feeder** e*Way and sends it to the **SOAPBabelFishClient** e*Way.
- **Out_Q** receives data from the **SOAPBabelFishClient** e*Way and sends it to the **Eater** e*Way.

Location of Schema Files

The completed BabelFish schema is included on the installation CD-ROM in the following location:

\samples\ewSOAP\BabelFish.zip

To do this implementation, you first need to unzip the **BabelFish.zip** file. The files listed in Table 2 are contained within this file.

Table 2 Sample Sender Schema Files

File Name	Description
BabelFish.zip	Export schema file
BabelFishRequest.dtd	Document Type Definition (DTD) that describes the BabelFish SOAP request
BabelFishResponse.dtd	DTD that describes the BabelFish SOAP response
Text.~in	Input file
Readme.txt	Information file

To use this sample schema, the SOAP e*Way must be installed, the sample schema must be installed, all of the necessary files and scripts must be located in the default location, and the **www.xmethods.net** Web site must be available.

Schema Implementation

To implement this sample schema, you can do one of the following operations:

- To import the sample schema zip file, which automatically creates the sample schema components, see the instructions provided in **"Sample Sender Schema: Automatic Implementation" on page 42**.
- To manually create each of the components required to use the sample schema, see the instructions provided in **"Sample Sender Schema: Manual Configuration" on page 44**.

5.2.2 Sample Sender Schema: Automatic Implementation

This section explains how to automatically implement the SOAP e*Way within a sample sender schema.

Installing and Configuring the Schema

To install and configure the BabelFish sample schema

1. Copy the file named **BabelFish.zip** from the SAMPLES directory in the install CD-ROM to your desktop or to a temporary directory, then unzip the file.
2. Start the e*Gate Enterprise Manager graphical user interface (GUI).
3. On the **Open Schema from Registry Host** dialog box, click **New**.
4. On the **New Schema** dialog box, click **Create from export**, and then click **Find**.
5. On the **Import from File** dialog box, browse to the directory that contains the sample schema, click **BabelFish.zip**, and then click **Open**.

   The sample schema is installed.

6 Configure the **Feeder** e*Way as follows:

A From e*Gate Enterprise Manager, display the properties of the **Feeder** e*Way, then click **Edit**. The e*Way Editor GUI appears; use this GUI to configure or modify an e*Way.

B In the **Goto Section** of the e*Way Editor, choose **Poller (inbound)** settings.

C For the **Poll Directory** parameter, specify the path name of the directory that contains the sample input data. This directory is named **\INDATA,** and it is located in the directory where you installed the sample schema.

7 Configure the **Eater** e*Way as follows:

A From the Enterprise Manager, display the properties of the **Eater** e*way, and then click **Edit**.

B In the **Goto Section** of the e*Way Editor, choose **Outbound (send)** settings.

C For the **OutputDirectory** parameter, specify the path name of the directory that contains the sample data. This directory is named **\data** and it is located in the directory in which you installed the sample schema.

Running the Schema

To run the BabelFish schema

1 From the command line prompt, enter:

```
stccb -rh hostname -rs schemaname -un username
-up user password -ln hostname_cb
```

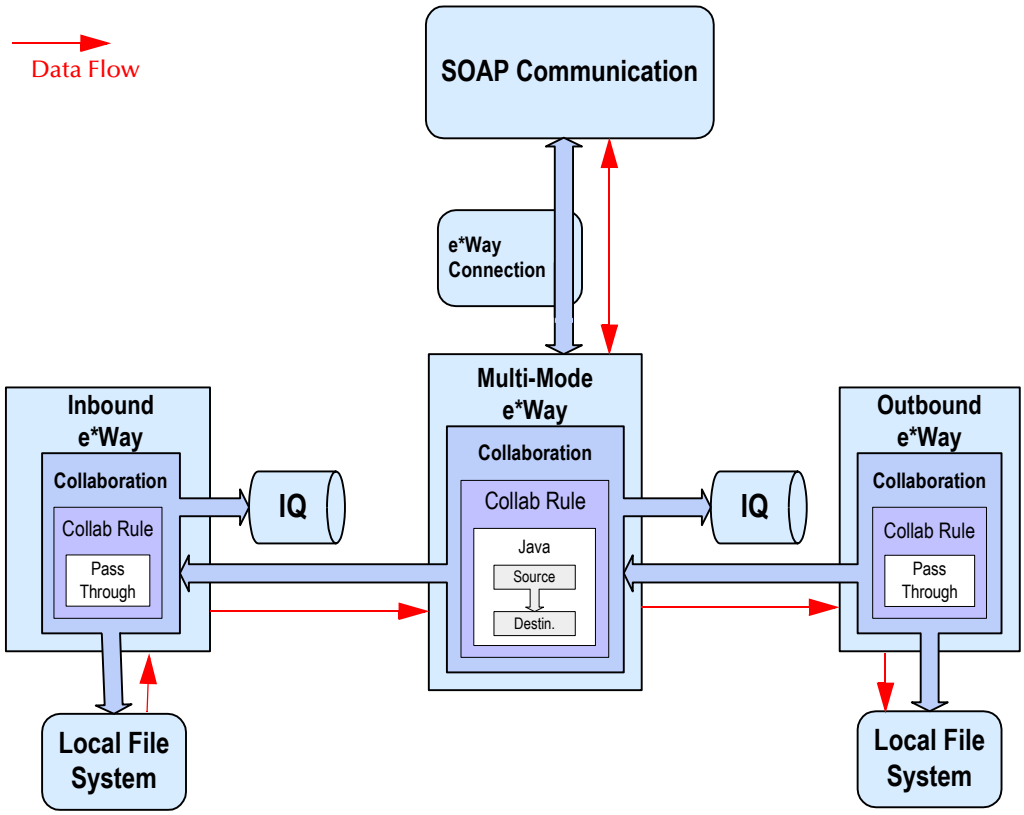
Substitute ***hostname***, ***username***, ***schemaname***, and ***user password*** as appropriate.

2 Change the input file name extension to **.fin**.

The schema components start automatically. When there are no more run-time messages, check the output file. If the schema is operating correctly, this file contains the text translated into French.

Figure 4 shows an overview diagram of the BabelFish schema and how it operates. The blue arrows show publication/subscription (pub/sub) relationships between the components. Red arrows show the actual flow of data.

Figure 4 BabelFish Schema Overview



5.2.3 Sample Sender Schema: Manual Configuration

This section explains how to configure the BabelFish sender schema manually in e*Gate, starting from the beginning.

Basic Implementation Steps

After you have located the SOAP service description, you must do the following steps:

1. Determine the SOAP endpoint URL.
2. Determine the format of the SOAP message.
3. Create a schema.
4. Create Event Types and Event Type Definitions (ETDs).
5. Create Collaboration Rules.
6. Create the e*Way Connection.
7. Add Intelligent Queues (IQs).
8. Add and configure e*Ways.
9. Create and configure Collaborations.
10. Test the schema.

The rest of this section explains each of the previous steps.

Note: *For complete explanations of procedures in steps 3 through 10, see* ***Creating an End-to-End Scenario with e*Gate Integrator*** *and the* ***e*Gate Integrator User's Guide****.*

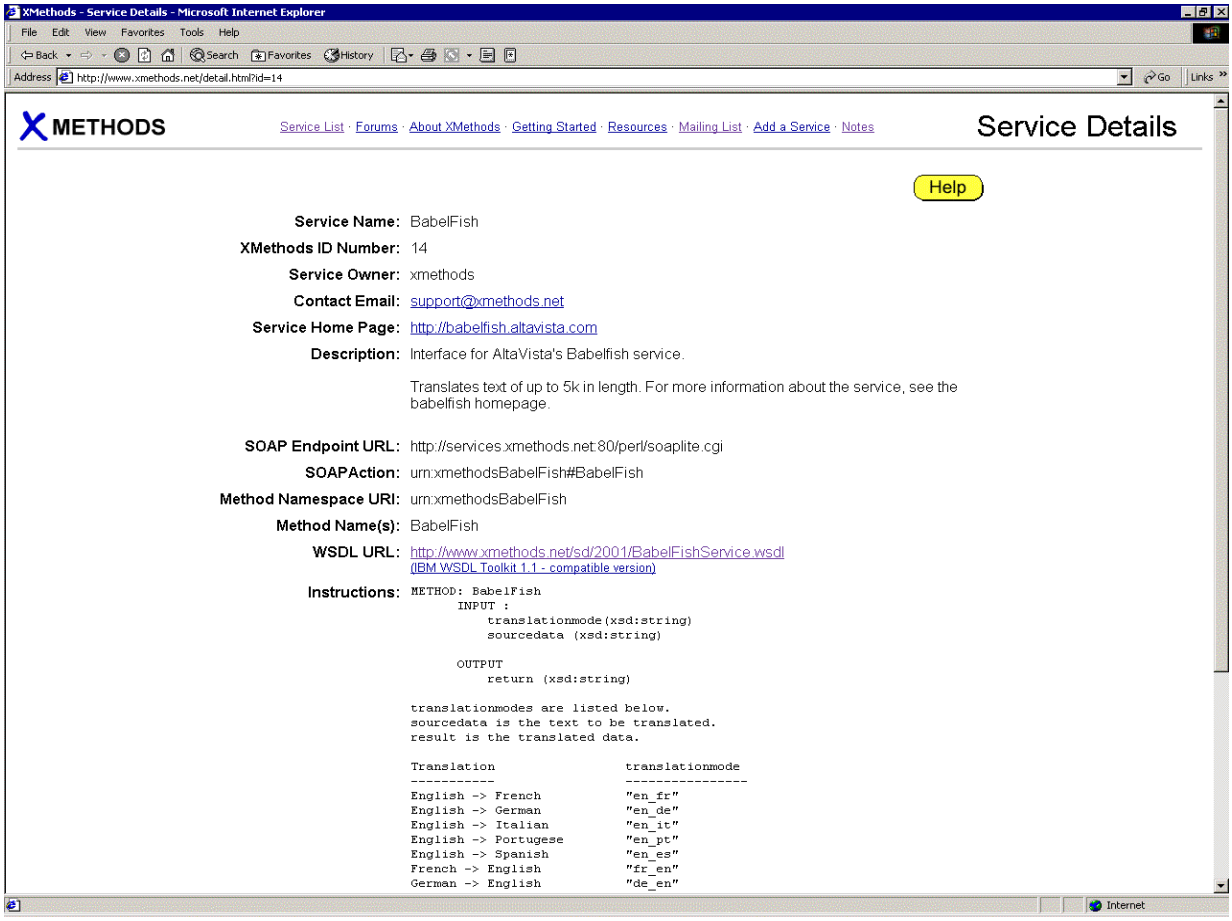
Step 1: Determine the SOAP Endpoint URL

Each service entry contains information describing the service. Find the service entry detail and locate the SOAP endpoint URL. For the BabelFish service, the URL is:

http://services.xmethods.net:80/perl/soaplite.cgi

For details, see **Figure 5 on page 46**.

Figure 5 BabelFish Service Entry Detail



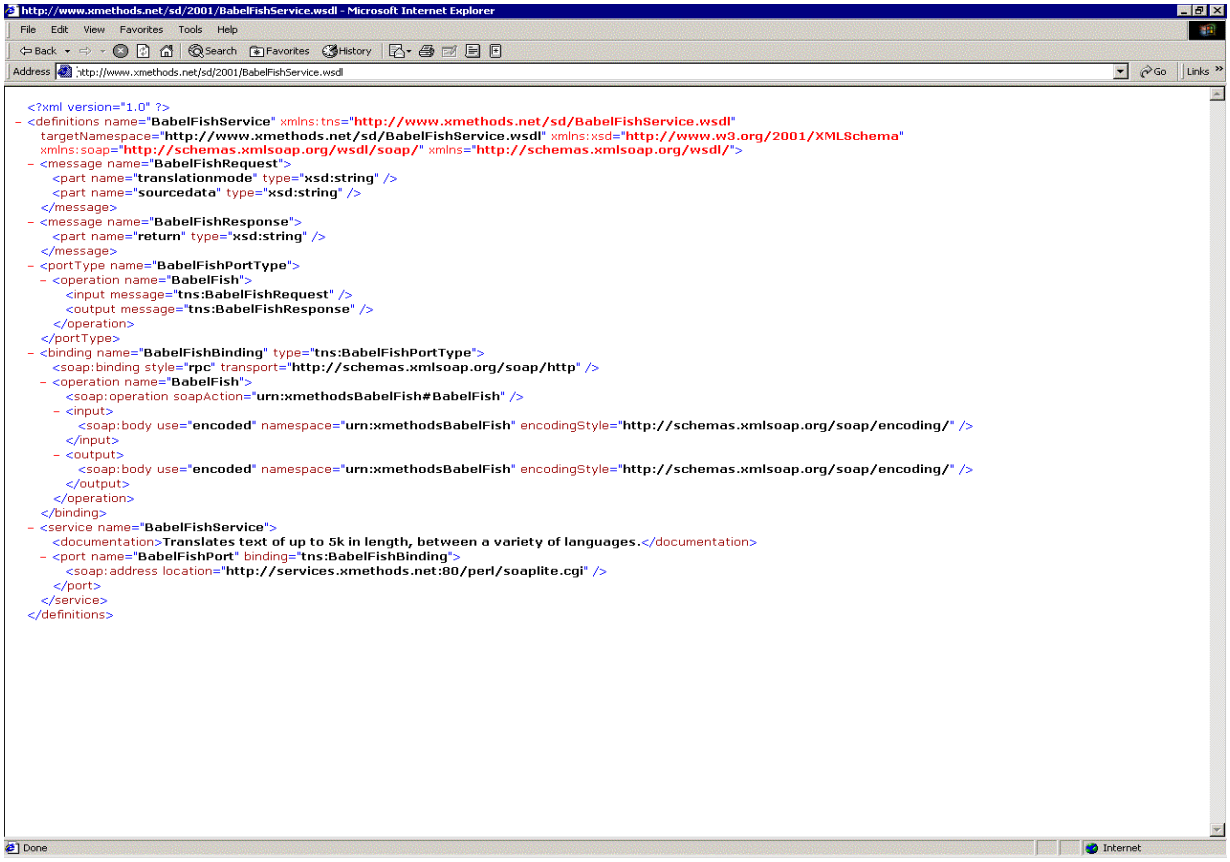
Step 2: Determine the Format of the SOAP Message

The service entry typically provides either an example of the SOAP message format or Web Services Definition Language (WSDL) URL. WSDL is an XML format for describing network services. WSDL is commonly used to describe the endpoints and message structure used with SOAP.

In this version of the SOAP e*Way and SOAP wizard, you have the option of describing the SOAP message as a byte stream BLOB, or as structured data. Using structured data allows the data to be marshalled and unmarshalled from the XML document and e*Gate Java Collaboration.

To use structured data, DTDs are required to describe any SOAP request header and bodies, and SOAP response header and bodies (see **Figure 6 on page 47**).

Figure 6 BabelFish DTD



The following examples illustrate DTDs that describe the SOAP message as documented in the BabelFish WSDL:

Example 1: BabelFishRequest.dtd

This DTD describes the SOAP request as follows:

```
<?xml encoding="UTF-8"?>

<!ELEMENT ns1:BabelFish (translationmode, sourcedata)>
<!ATTLIST ns1:BabelFish SOAP-ENV:encodingStyle CDATA #REQUIRED>
<!ATTLIST ns1:BabelFish xmlns:SOAP-ENV CDATA #FIXED "http://
schemas.xmlsoap.org/soap/envelope/">
<!ATTLIST ns1:BabelFish xmlns:ns1 CDATA #FIXED
"urn:xmethodsBabelFish">

<!ELEMENT translationmode (#PCDATA)>
<!ATTLIST translationmode xsi:type CDATA #FIXED "xsd:string">
<!ATTLIST translationmode xmlns:xsi CDATA #FIXED "http://www.w3.org/
1999/XMLSchema-instance">

<!ELEMENT sourcedata (#PCDATA)>
<!ATTLIST sourcedata xsi:type CDATA #FIXED "xsd:string">
<!ATTLIST sourcedata xmlns:xsi CDATA #FIXED "http://www.w3.org/1999/
XMLSchema-instance">
```

Example 2: BabelFishResponse.dtd

This DTD describes the SOAP response as follows:

```
<?xml encoding="UTF-8"?>

<!ELEMENT namesp1:BabelFishResponse ( return ) >
<!ATTLIST namesp1:BabelFishResponse SOAP-ENV:encodingStyle CDATA
#FIXED "http://schemas.xmlsoap.org/soap/encoding">
<!ATTLIST namesp1:BabelFishResponse xmlns:SOAP-ENV CDATA #FIXED
"http://schemas.xmlsoap.org/soap/envelope/">
<!ATTLIST namesp1:BabelFishResponse xmlns:namesp1 CDATA #FIXED
"urn:xmethodsBabelFish">

<!ELEMENT return ( #PCDATA ) >
<!ATTLIST return xsi:type CDATA #FIXED "xsd:string">
<!ATTLIST return xmlns:xsi CDATA #FIXED "http://www.w3.org/1999/
XMLSchema-instance">
```

Note: *The DTDs shown in the previous examples were created manually from information contained in the WSDL entry.*

Step 3: Create a Schema

Before creating a schema, first verify that you have the correct e*Gate installation and that it is operating correctly.

Verifying the e*Gate Installation

You can run this schema on a single machine. Before beginning the configuration process, you must verify that you have all the required software installed on the target machine.

Check the following e*Gate system components:

- Registry Host
- Participating Host
- GUIs
  - e*Gate Enterprise Manager
  - e*Gate Monitor

You can install all the software components shown in the previous list on the machine that runs this schema. See the *e*Gate Integrator Installation Guide* for instructions on how to install the e*Gate components and for e*Gate system requirements.

To create a new schema

1. Start the e*Gate Enterprise Manager and log in as **Administrator** (or another user with administrator privileges) to the appropriate Registry Host.
2. In the **Open Schema on Registry Host** dialog box, click **New**.

3 In the **Enter New Schema Name** box, type **BabelFish**, and then click **Open**.

The Enterprise Manager opens and displays the new **BabelFish** schema.

4 At the bottom of the Navigator (left) pane, click the **Components** tab.

You perform all configuration steps in this pane, on the **Components** tab.

Note: *When you create a new schema, by default, e*Gate automatically creates a Control Broker for the schema. The default name is* ***host-name_cb****, where* ***host-name*** *is the logical name of the current host machine. For this example, use these default Control Broker name and its default settings.*

Step 4: Create Event Types and Event Type Definitions

In this step, you create Event Types and Event Type Definitions (ETDs) that the e*Gate system uses to transport data.

Creating Event Types

An Event Type is a class of Events (packages of data) with a common data structure. The e*Gate system packages data within Events and categorizes them into Event Types. What these Events have in common defines the Event Type and comprises the ETD.

The following procedures show how to create an ETD using the an ETD wizard (see **"Using the SOAP Wizard" on page 50**).

To create Event Types

1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Enterprise Manager's Navigator pane (**Components** tab).

2 On the palette, click the **Create a New Event Type** button.

A dialog box opens allowing you to enter the name of the new Event Type.

3 Enter the name of the Event Type. For the purpose of this sample, the SOAP Event Type is named **SOAP_BabelFish_Event**.

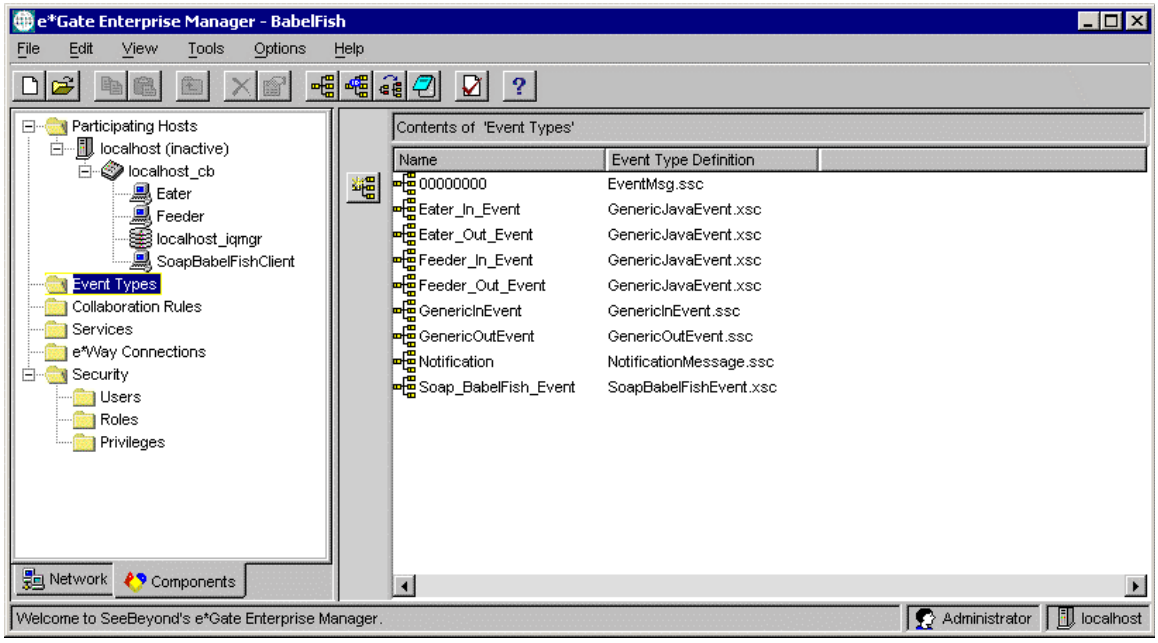
4 Click **OK.** The dialog box closes, and e*Gate saves the name of your new Event Type.

Using these steps, create the following Event Types:

- **SOAP_BabelFish_Event:** This Event Type contains the request data, the response data (if any), and the methods used to manipulate the data.
- **Feeder_In_Event:** This Event Type contains raw data from the input file.
- **Feeder_Out_Event:** This Event Type contains raw data from the input file.
- **Eater_In_Event:** This Event Type contains the translated data.
- **Eater_Out_Event:** This Event Type contains the translated data.

When you have finished, the Enterprise Manager GUI shows all of your created Event Types (see **Figure 7 on page 50**).

Figure 7 e*Gate Enterprise Manager: Event Types



Defining the SOAP ETD

Next, you must define the **SOAP_BabelFish_Event** Event Type you have created, that is, create its *Event Type Definition*. To define this Event Type, create an ETD file named **SoapBabelFishEvent.xsc** that describes the SOAP Message.

You can use any of the following methods to create this ETD:

- Using the ETD **\eGate\client\etd\ewsoap\SOAPSimple.xsc**
- Modifying a copy of the ETD **\eGate\client\etd\ewsoap\SOAPSimple.xsc**
- Using the SOAP wizard to create an ETD describing a byte stream BLOB
- Using the SOAP wizard to create an ETD describing a structured XML document

Use the Enterprise Manager's ETD Editor GUI to create and modify ETDs. The ETD Editor has convenient wizard features that help you to create ETDs. These sample procedures use the SOAP wizard.

The rest of this section (Step 4) explains how to use the SOAP wizard to create the **SoapBabelFishEvent.xsc** ETD file, a structured XML document.

Using the SOAP Wizard

The SOAP wizard takes a DTD file and converts it to an **.xsc** file that contains the following elements:

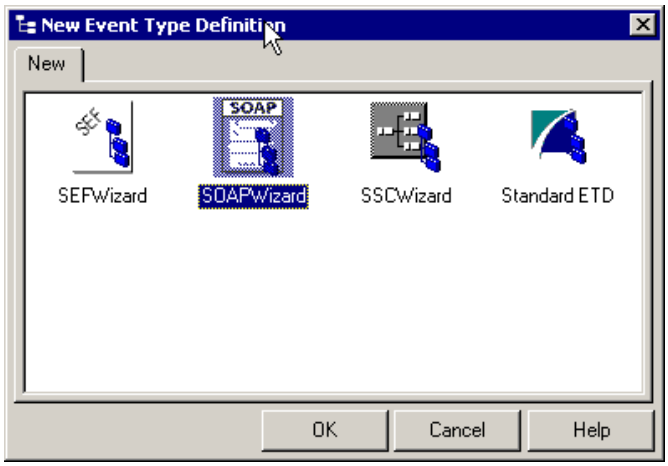
- **SOAP request header element**: SOAP header of the request message
- **SOAP request body element**: SOAP body of the request message
- **SOAP response header element**: SOAP header of the response message
- **SOAP response body element**: SOAP body of the response message

You can use the SOAP wizard to create a generic ETD using well-formed XML but no header or body format, or you can create an ETD from a specific DTD file. If you choose the latter, the resulting ETD will contain precise format, syntax, and semantics for the SOAP request and response header and body.

To convert a SOAP DTD to an e*Gate ETD using the wizard

1 To access the SOAP wizard, select the **New** option in the ETD Editor's **File** menu. The New Event Type Definitions window appears, displaying all installed ETD wizards (see Figure 8).

Figure 8 New Event Type Definition Window



2 Click the **SOAP Wizard** icon.

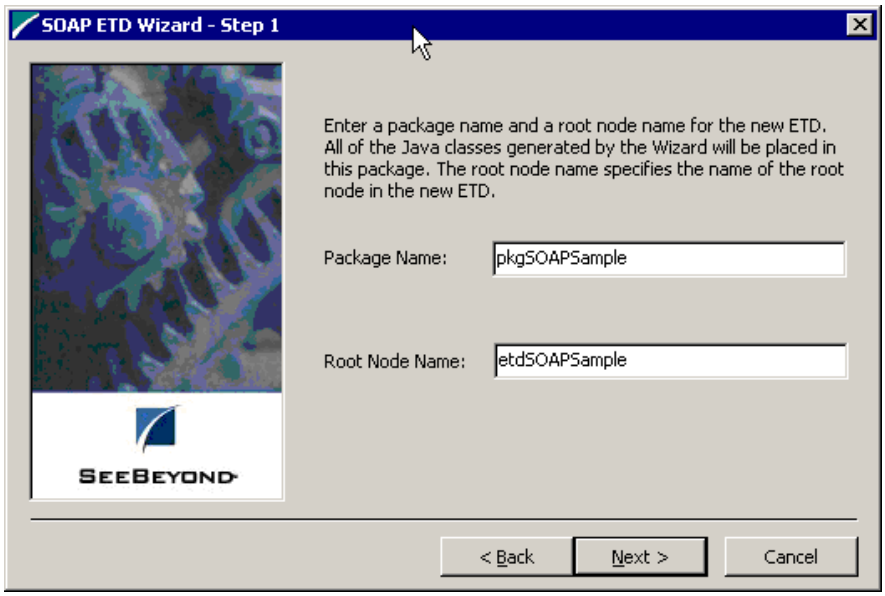
3 Review the **SOAP ETD Wizard Introduction** page, then click **Next.**

4 On **SOAP ETD Wizard - Step 1** (see **Figure 9 on page 52**), enter the following text:

- The Package Name for the container in which the wizard places the generated Java classes
- The root name of the ETD

5 Click **Next** to continue.

Figure 9 SOAP ETD Wizard - Step 1 Window



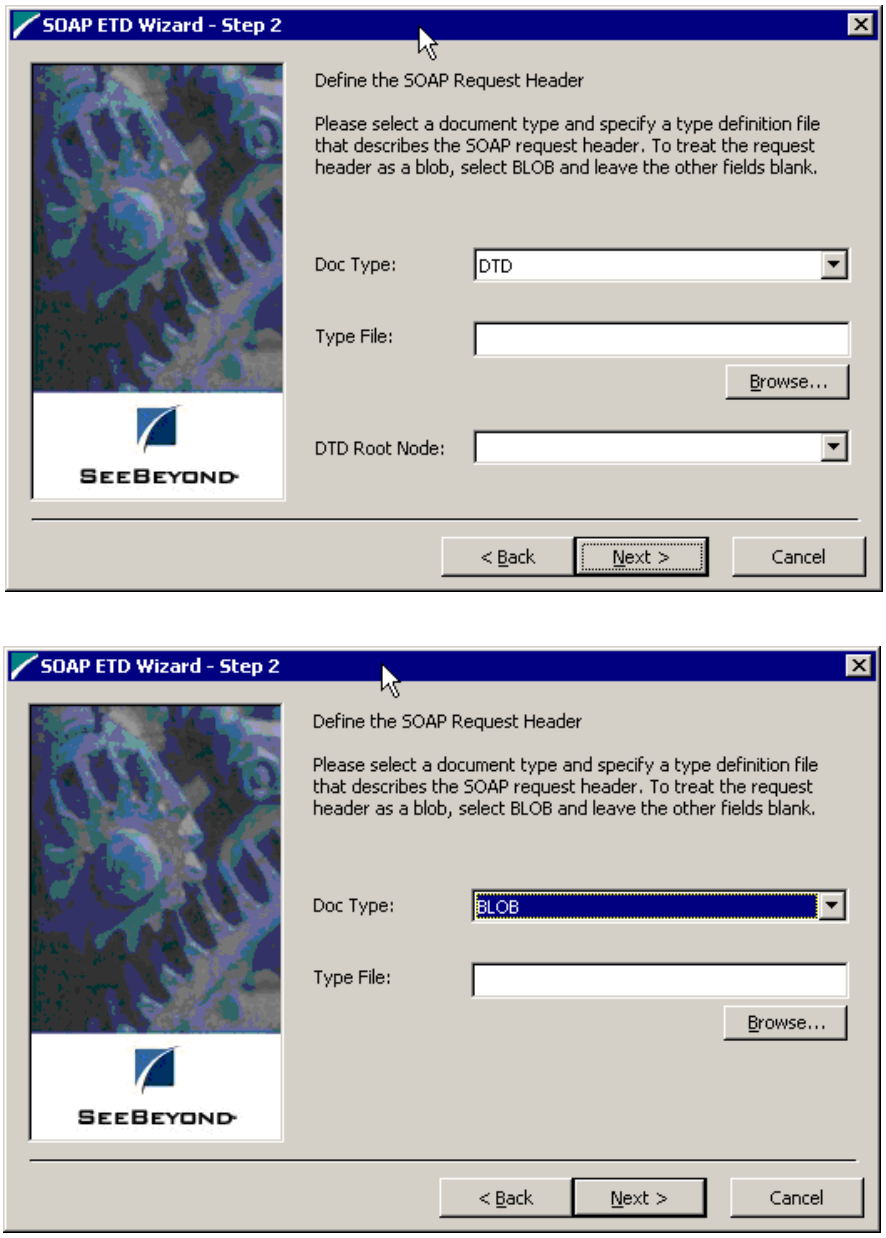
6 On **SOAP ETD Wizard - Step 2** (see **Figure 10 on page 53**), do one of the following operations to describe the contents of the SOAP request header element:

- To create a generic SOAP request header element, choose **BLOB** as the document type and continue to the next step.
- To create the SOAP request header element from a DTD file, choose **DTD** as the document type, and enter the name of the type definition file. At this point, all top level elements defined in the DTD file will be listed in the **DTD root node** pull down list. Select the element you want to use for the SOAP request header.

If you do not know the name of the type file, click **Browse** to navigate to the appropriate file. All the elements in the DTD file are listed as possible root node names for this component.

7 Click **Next** to continue.

Figure 10 SOAP ETD Wizard - Step 2 Window (SOAP Request Header Element)



8 On the next **SOAP ETD Wizard - Step 2**, do one of the following steps to describe the contents of the SOAP request body element:

- To create a generic SOAP request body element, choose **BLOB** as the document type and continue to the next step. A type file is not required.
- To create the SOAP request body element from a DTD file, choose **DTD** as the document type, enter the type file, and choose a DTD root node name from the list.

If you do not know the name of the type file, click **Browse** to navigate to the appropriate file. All the elements in the DTD file are listed as possible root node names for this component.

9 Click **Next** to continue.

10 On the next **SOAP ETD Wizard - Step 2**, do one of the following to describe the contents of the SOAP response header element:

- To create a generic SOAP response header element, choose **BLOB** as the document type and continue to the next step.
- To create the SOAP response header element from a DTD file, choose **DTD** as the document type, enter the type file, and choose a DTD root node name from the list.

If you do not know the name of the type file, click **Browse** to navigate to the appropriate file. All the elements in the DTD file are listed as possible root node names for this component.

11 Click **Next** to continue.

12 On the next **SOAP ETD Wizard - Step 2**, do one of the following operations to describe the contents of the SOAP response body element:

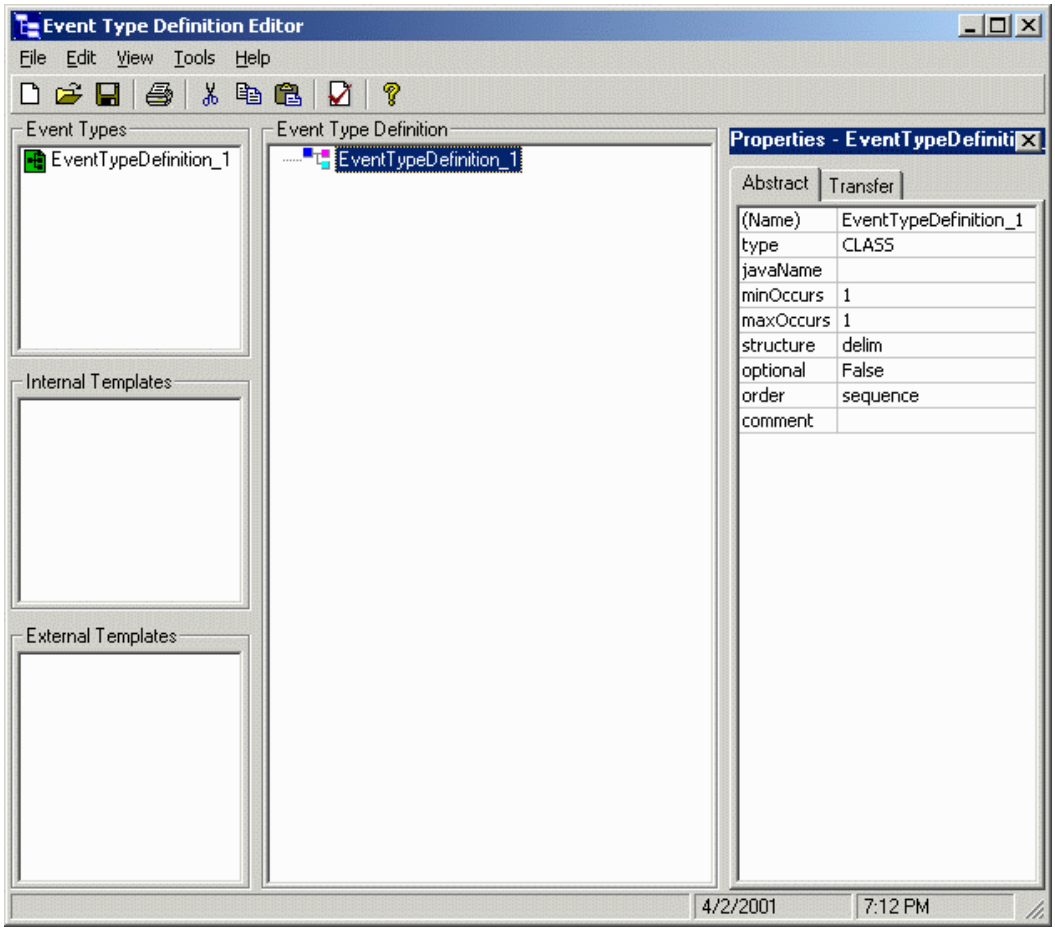
- To create a generic SOAP response body element, choose **BLOB** as the document type and continue to the next step.
- To create the SOAP response body element from a DTD file, choose **DTD** as the document type, enter the type file, and choose a DTD root node name from the list.

If you do not know the name of the type file, click **Browse** to navigate to the appropriate file. All the elements in the DTD file are listed as possible root node names for this component.

13 Click **Finish** when you are done with the wizard.

The SOAP ETD appears in the ETD Editor Main window as shown in **Figure 11 on page 55**.

Figure 11 ETD Editor: SOAP ETD



14 The title of the ETD, **EventTypeDefinition_1,** in the ETD you just created is an invalid name and must be changed. Change the name using either of the following steps:

- Click on **EventTypeDefinition_1** twice in the **Event Type Definition** pane, type the new name (**BabelFish**), and press ENTER.
- Highlight **EventTypeDefinition_1** under the **Abstract** tab in the **Properties** pane, type a new name **BabelFish**, and press ENTER.

15 Close the ETD Editor and save the ETD under the name **BabelFish.xsc.** This is your new ETD file.

Defining the Additional ETDs

As explained previously, the BabelFish schema also uses the following Event Types:

- **Feeder_In_Event**
- **Feeder_Out_Event**
- **Eater_In_Event**
- **Eater_Out_Event**

Additionally, you must define each of these types, creating/defining an ETD for each one.

To define the "Feeder/Eater" ETDs

1 In Enterprise Manager, on the toolbar or **Tools** menu, click the **ETD Editor** button in the Toolbar.

Note: *Be sure you have set your default editors to Java. Use the Options menu.*

The ETD Editor Main window appears, with all panes empty.

2 In the ETD Editor, on **File** menu, click **New**.

The **New Event Type Definition** dialog box displays icons for the ETD wizards (see **Figure 8 on page 51**).

3 Select the **Standard ETD** wizard icon and click **OK**.

The Standard ETD wizard launches displaying the **Standard ETD Wizard - Introduction** dialog box (see Figure 12). It informs you that you must specify a "Package Name" where all the Java source files are generated.

Figure 12 Standard ETD Wizard: Introduction

Note: *For complete details on how to use all the ETD wizards, see the* ***e*Gate Integrator User's Guide****.*

4 Click **Next**.

The **Standard ETD Wizard - Step 1** dialog box appears.

5 Enter a **Package Name** in the text box (**pkgSOAPSample**). This is where the ETD Editor places all the generated Java classes associated with this ETD. This name must be a legal Java package name, such as **custominPackage**.

6 When you are finished, click **Next**.

7 The **Standard ETD Wizard - Step 2** dialog box opens, requesting that you confirm that the **Package Name** you entered is correct. If it is not correct, click **Back** to change the name, or, if correct, click **Finish** to generate the ETD.

The ETD you just created appears in the **Event Type Definition** pane in the ETD Editor, in the same way as the SOAP ETD created previously.

8 The title of the ETD, **EventTypeDefinition1,** in the ETD you just created is an invalid name and must be changed. Change the name using either of the following steps:

- Click on **EventTypeDefinition1** twice in the **Event Type Definition** pane, type the new name (**Feeder_In_Event**), and press ENTER.
- Highlight **EventTypeDefinition1** under the **Abstract** tab in the **Properties** pane, type a new name **Feeder_In_Event**, and press ENTER.

9 Repeat these procedures to create the rest of the ETDs: **Feeder_Out_Event, Eater_In_Event,** and **Eater_Out_Event.**

Note: *See the BabelFish schema sample for details on how to create these ETDs. Open the ETDs given in the sample, and set up each of the remaining ETDs in the same way as they are set up in the sample.*

Step 5: Create Collaboration Rules

In the e*Gate system, Events become subject to business logic via processing, transformation, or verification. e*Gate uses the following components to govern these operations:

- **Collaboration** is the necessary, configurable component of an e*Way that determines its operation; that is, the logical moving and transformation of Events.
- **Collaboration Rules** are the program logic that instructs a Collaboration how to execute the business logic required to support e*Gate's data transformation and routing.
- **Collaboration Service** is the program that defines the structure and operation of a Collaboration Rule's basic Event-handling processes. For example, Java Collaborations use the Java Collaboration Service.
- **Collaboration Rules Script** contains the specific operations (written in the Monk programming language) that are used to govern Event-transformation processes within a Collaboration.
- **Business Rules** are the Java source code that creates the output Events that are a result of the Java Collaboration.

Collaboration **.class** files (Java) and Collaboration Rules scripts (Monk) are necessary if you want to have any data transformed and/or verified in some way as it passes through a Collaboration.

You must create Collaboration Rules before you create the Collaborations that use them. For details on how to create the BabelFish Collaborations, see **"Step 9: Create and Configure Collaborations" on page 67**.

For the BabelFish schema, you must create the following Collaboration Rules:

- **Feeder:** Pass Through Collaboration Rule (uses the Pass Through Collaboration Service) and associated with the **Feeder** e*Way.
- **Eater:** Pass Through Collaboration Rule and associated with the **Eater** e*Way.
- **SOAPBabelFishClient:** associated with the **SOAPBabelFishClient** e*Way and is used to perform the transformation process, send the Event to the SOAP service, and receive a response from the SOAP service.

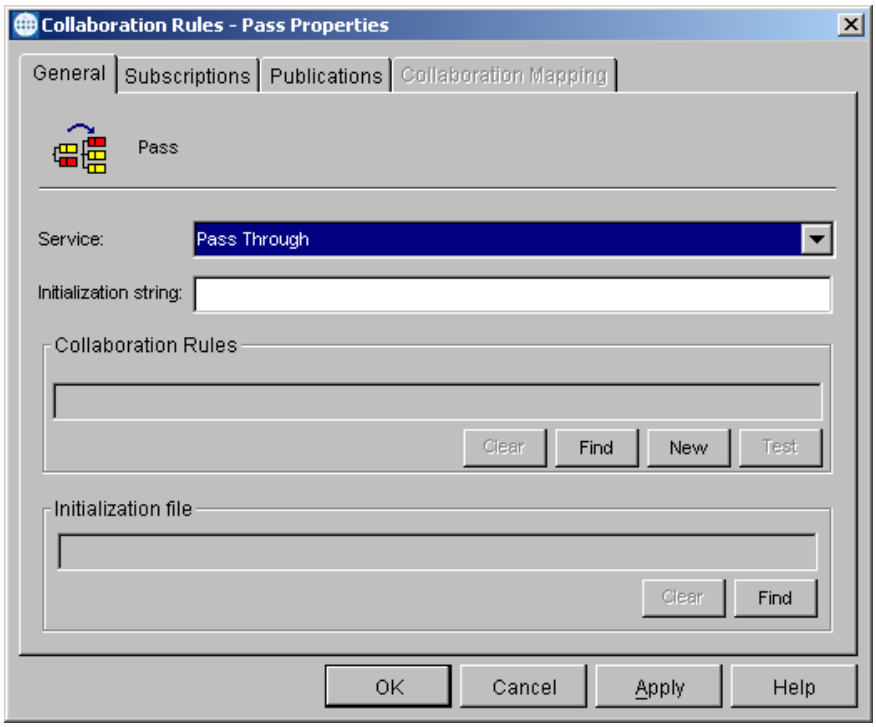
The following pseudo-code helps to explain the actions of the **SOAPBabelFishClient** e*Way Collaboration/Collaboration Rules:

```
Set translation mode to en_fr
Populate the data source - an English sentence
Send SOAP request to the SOAP server and unmarshal the response into
SOAPResponse
Get the translated French sentence
```

To create Pass Through Collaboration Rules components

1. Select the Navigator pane's **Components** tab in the e*Gate Enterprise Manager.
2. In the **Navigator**, select the **Collaboration Rules** folder.
3. On the palette, click the **Create New Collaboration Rules** button.
4. Enter the name of the new Collaboration Rule Component, **Feeder**, then click **OK**.
5. Double-click the new Collaboration Rules Component icon. The **Collaboration Rules Properties** dialog box opens (see **Figure 13 on page 59**).

Figure 13 Collaboration Rules Properties Dialog Box: Pass Through



6 The **Service** field defaults to **Pass Through**; accept the default.

7 Go to the **Subscriptions** tab. Select **GenericInEvent** under **Available Input Event Types**, and click the right arrow to move it to **Selected Input Event Types**. The box under **Triggering Event** should be checked.

8 Go to the **Publications** tab. Select **GenericInEvent** under **Available Output Event Types**, and click the right arrow to move it to **Selected Output Event Types**. The Radio button under **Default** is enabled.

9 Click **OK** to close the **Collaboration Rules, Pass Properties** window.

10 Repeat this same procedure for the **Eater** Collaboration Rule.

The **SOAPBabelFishClient** Collaboration Rule uses the Java Collaboration Service. The procedures for creating it are different from those used for the Pass Through Collaboration Rules.

To create and edit the Java Collaboration Rules component

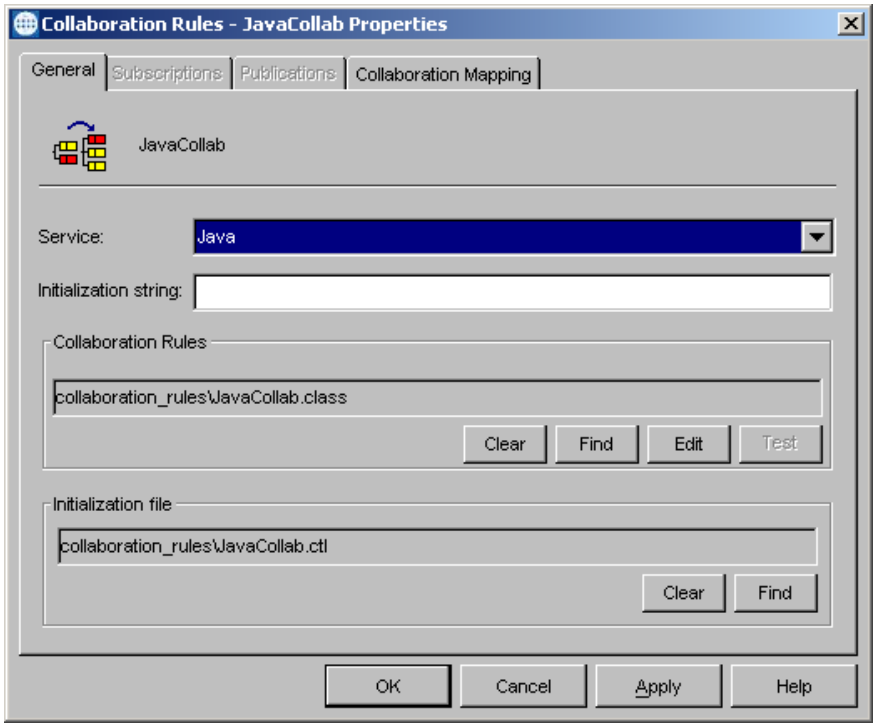
1 Use the Enterprise Manager to create the new Java Collaboration Rules component in the same way as you did the Pass Through Collaboration Rules components.

2 Double-click the new Collaboration Rules component icon to edit its properties.

The **Collaboration Rules Properties** dialog box opens (see Figure 13).

3 From the **Service** field drop-down box, select Java. The **Collaboration Mapping** tab is now enabled, and the **Subscriptions** and **Publications** tabs are disabled (see **Figure 14 on page 60**).

Figure 14 Collaboration Rules Properties Dialog Box: Java



4 In the **Initialization string** field, enter any required initialization string for the Collaboration. If none is needed, you can skip this step.

5 Select the **Collaboration Mapping** tab.

6 Using the **Add Instance** button, create instances to coincide with the Event Types as follows:

- In the **Instance Name** column, enter **In** for the instance name.
- Click **Find**, navigate to **etd\BabelFish.xsc**, double-click to select. **BabelFish.xsc** is added to the **ETD** column of the instance row.
- In the **Mode** column, select **In** from the drop–down menu available.
- In the **Trigger** column, click the box to enable trigger mechanism.

7 Repeat the actions listed under step 6 using the following values:

- Instance Name: **Out**
- ETD: **BabelFish.xsc**
- Mode: **Out**

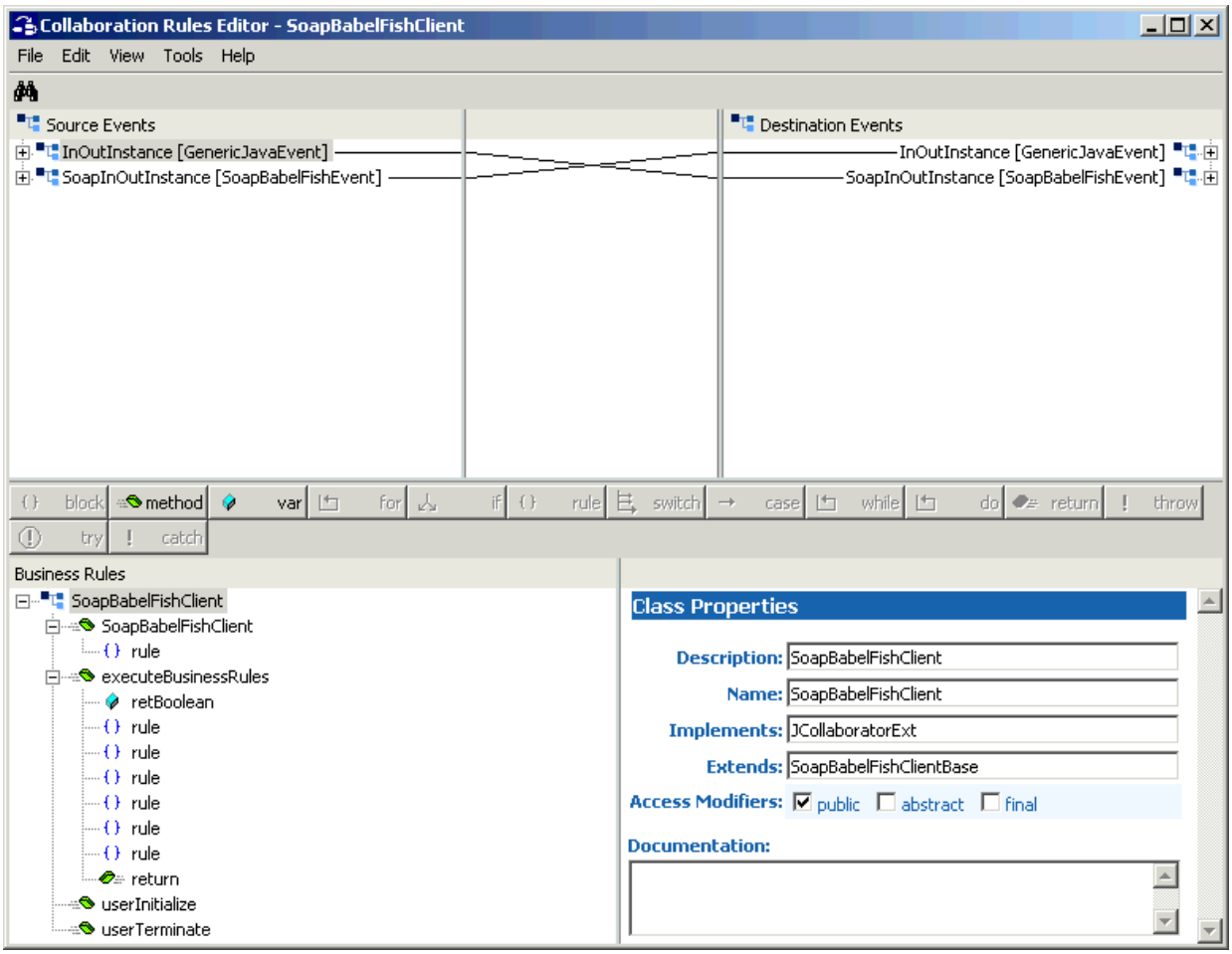
Note: *At least one of the ETD instances used by the Collaboration must be checked as the trigger.*

For specific information on creating and configuring Collaboration Rules, see the ***e*Gate Integrator User's Guide****.*

8 Select the **General** tab, under the Collaboration Rule box, select **New**. The Collaboration Rules Editor GUI opens (see **Figure 15 on page 61**).

Note: *The example in the figure shows the opened* ***SoapBabelFishClient.xpr*** *file from the sample.*

Figure 15 Collaboration Rules: Collaboration Rules Editor



9 Expand the Collaboration Rules Editor to full size, expanding the Source and Destination Events panes as well, then create the Collaboration Rule.

Using the Collaboration Rules Editor

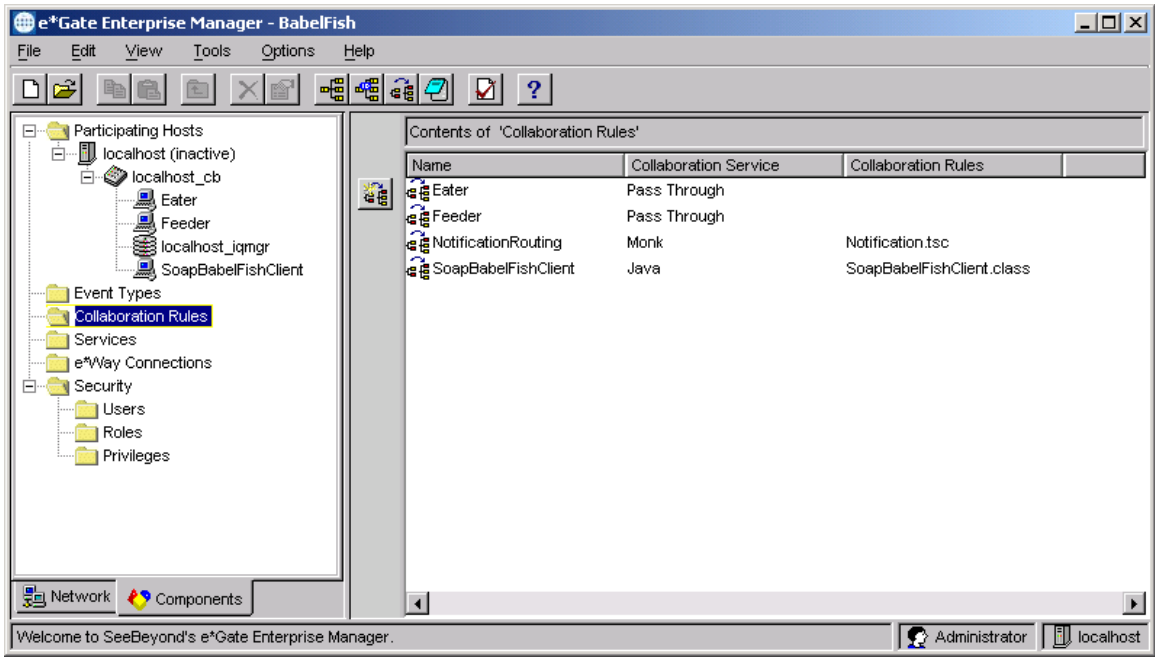
The next part of this step is to define the business logic using the Collaboration Rules Editor. The Java Collaboration Rules Editor is the GUI used to create and modify Java Collaboration Rules (Business Rules).

A Java Collaboration Rule is created by designating one or more source ETDs and one or more destination ETDs then setting up rules governing the relationship between fields in the two ETDs. Use the Collaboration Rules Editor to tell e*Gate how you want data taken from the source ETD, then manipulated and placed in the destination ETD.

Note: *To create the Collaboration Rule for SoapBabelFishClient, open the .xpr file from the sample and configure your new Collaboration Rule in the same way. For complete information on creating Collaboration Rules using the Java Collaboration Rules Editor see the* ***e*Gate Integrator User's Guide****.*

Figure 16 on page 62 shows the created Collaboration Rules in the Enterprise Manager GUI.

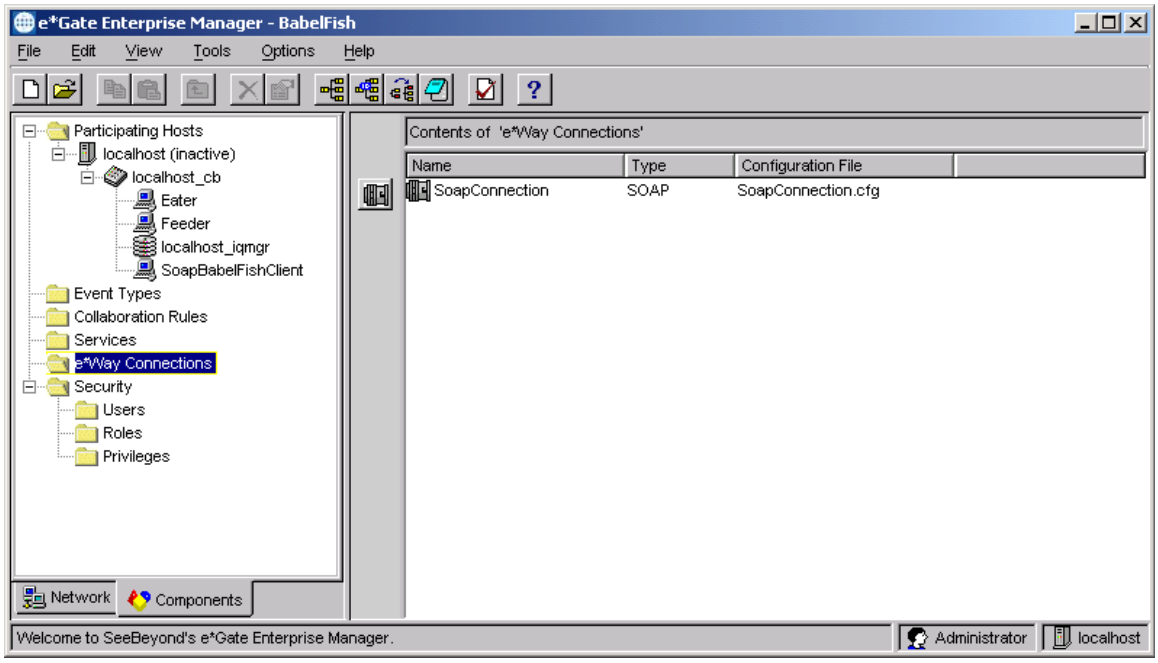
Figure 16 e*Gate Enterprise Manager with Collaboration Rules



Step 6: Create the e*Way Connection

See **Chapter 4** for complete information on how to configure the SOAP e*Way Connection. **Figure 17 on page 63** shows the created e*Way Connection **SoapConnection** in the Enterprise Manager Main window.

Figure 17 e*Gate Enterprise Manager with e*Way Connection



Step 7: Create Intelligent Queues

The next step in setting up the BabelFish schema is to create the IQs. IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another.

IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs and handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

For the BabelFish schema, you must create the following IQs:

- **In_Q** receives data from the **Feeder** e*Way and sends it to the **SOAPBabelFishClient** e*Way.
- **Out_Q** receives data from the **SOAPBabelFishClient** e*Way and sends it to the **Eater** e*Way.

To create and modify the IQs

1. Select the **Navigation** pane's **Components** tab.
2. Open the host where you want to create the IQ.
3. Open the desired **Control Broker**.
4. Select the desired **IQ Manager**.
5. On the palette, click the **Create a New IQ** button.
6. Enter the name of the new IQ (in this case, **In_Q**), then click **OK.**
7. Double-click the new IQ's icon in the **Navigation** pane to edit its properties.

The **IQ Properties** dialog box appears.

8 On the **General** tab, specify the **Service** and the **Event Type Get Interval**. Configure these settings as follows:

- The **STC_Standard** IQ Service provides sufficient functionality for most applications. If specialized services are required, you can create custom IQ Service **.dll** files.
- The default **Event Type Get Interval** of 100 ms is satisfactory for the purposes of this sample implementation.

9 On the **Advanced** tab, be sure that **Simple publish/subscribe** is checked under the **IQ behavior** section.

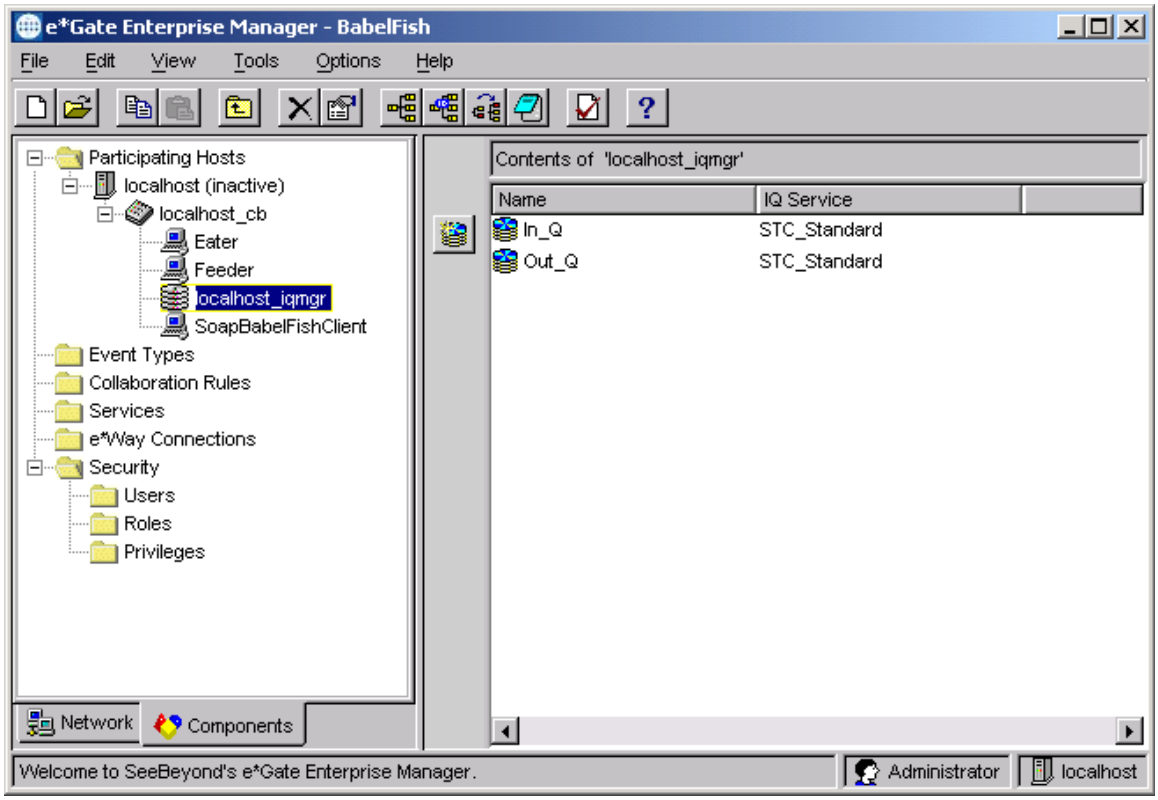
10 Click **OK** to close the **IQ Properties** dialog box.

11 For this schema, repeat this procedure to create an additional IQ (**Out_Q**).

The IQs you have created appear in the Enterprise Manager Main window (see **Figure 18 on page 64**)

Note: *For more details on this procedure, see* ***Creating an End-to-End Scenario with e*Gate Integrator*** *and/or the* ***e*Gate Integrator User's Guide****.*

Figure 18 e*Gate Enterprise Manager with IQs



Step 8: Add and Configure e*Ways

For the BabelFish schema, you must create the following e*Ways:

- **Feeder** receives text from an external source, applies Pass Through Collaboration Service, and publishes the information to an IQ that stores inbound data.
- **Eater** receives the outbound message from an IQ and publishes it to a file; it also uses the Pass Through Collaboration Service.
- **SOAPBabelFishClient** applies extended Java Collaboration Rules to an inbound Event to perform the desired business logic. In this case, the e*Way translates the inbound Event into a SOAP message, sends the SOAP message to a SOAP service, receives a translated text response from the SOAP service, and publishes the response to an IQ.

  This e*Way uses the executable file **stceway.exe**, causing the e*Way to become a Multi-Mode e*Way. For details on the Multi-Mode e*Way, see **Chapter 3**.

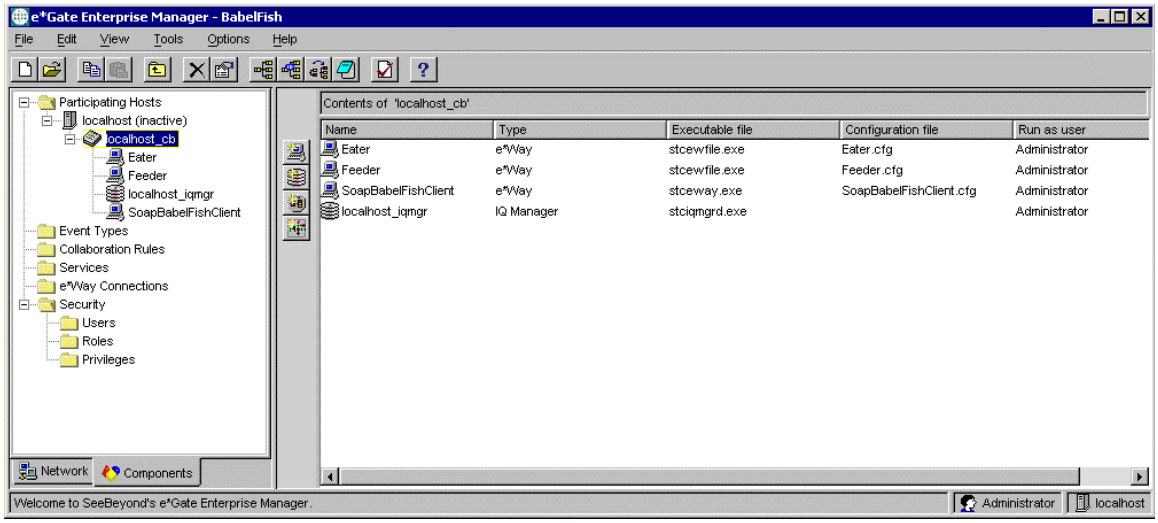
To create new e*Ways

1. In the **Navigator** pane (**Components** tab), select the desired Control Broker.
2. On the Palette, click the **Create a New e*Way** button.

   The **New e*Way Component** dialog box appears.
3. Enter the desired name (**Feeder**) for the new e*Way and click **Apply** to enter it into the system. The new name and an e*Way icon appear in both panes.
4. Name additional e*Ways as needed (**Eater** and **SOAPBabelFishClient**). Click **Apply** after you name each one.
5. When you are finished, click **OK** to close the dialog box.

   The new e*Way icons appear in the Enterprise Manager Main window as shown in Figure 19.

Figure 19 e*Gate Enterprise Manager with e*Ways



To configure the Feeder e*Way

1. From e*Gate Enterprise Manager, double-click on the **Feeder** e*Way icon to display the properties of the e*Way.

   The **e*Way Properties** dialog box appears.
2. Select the executable file **stcewfile.exe**.
3. Click **New**.

   The e*Way Editor GUI appears.
4. In the **Goto Section** of the e*Way Editor, choose the **Poller (inbound)** settings.
5. For the **Poll Directory** parameter, specify the path name of the directory that contains the sample input data. This directory is named **\INDATA** and it is located in the directory where you installed the sample schema.
6. Close the e*Way Editor and save the e*Way configuration file (**Feeder.cfg**).

To configure the Eater e*Way

1. From e*Gate Enterprise Manager, double-click on the **Eater** e*Way icon to display the properties of the e*way, then click **New**.

   The **e*Way Properties** dialog box appears.
2. Select the executable file **stcewfile.exe**.
3. Click **New**.

   The e*Way Editor GUI appears.
4. In the **Goto Section** of the e*Way Editor, choose **Outbound (send)** settings.
5. For the **OutputDirectory** parameter, specify the path name of the directory that contains the sample data. This directory is named **\data** and it is located in the directory in which you installed the sample schema.
6. Close the e*Way Editor and save the e*Way configuration file (**Eater.cfg**).

To configure the SOAPBabelFishClient e*Way

1. From e*Gate Enterprise Manager, double-click on the **SOAPBabelFishClient** e*Way icon to display the properties of the e*Way, then click **New.**

   The **e*Way Properties** dialog box appears.
2. Select the executable file **stceway.exe** to create a Multi-Mode e*Way.
3. Click **New**.

   The e*Way Editor GUI appears.
4. Configure this e*Way. For details on how to configure the Multi-Mode e*Way, see **Chapter 3**.
5. Close the e*Way Editor and save the e*Way configuration file (**SOAPBabelFishClient.cfg**).

Step 9: Create and Configure Collaborations

You must create the following Collaborations:

- **Feeder:** associated with the **Feeder** e*Way and is used for receiving input Events into e*Gate; uses the Feeder Collaboration Rule you created previously.
- **Eater:** associated with the **Eater** e*Way and is used for sending Events out of e*Gate; uses the Eater Collaboration Rule you created previously.
- **SOAPBabelFishClient:** associated with the **SoapBabelFishClient** e*Way and is used to perform the transformation process, send the Event to the SOAP service, and receive a response from the SOAP service; uses the SOAPBabelFishClient Collaboration Rule you created previously.

To create the Collaborations

1. In the **Navigator** pane (**Components** tab), select the desired Control Broker.
2. Select the desired e*Way component (**Feeder**).
3. On the Palette, click the **Create a New Collaboration** button.

   The **New Collaboration Component** dialog box appears.
4. Enter the desired name (**Feeder**) for the new Collaboration and click **OK** to enter it into the system. The new name and a **Collaboration** icon appear in the Editor (right) pane.
5. Repeat these procedures to create the **Eater** and **SOAPBabelFishClient** Collaborations. Click **OK** after you name each one.

To configure the Collaborations

1. Double-click on the icon for the desired Collaboration (for this example, choose **Feeder** first).

   The **Collaboration Properties** dialog box appears.
2. Configure the properties for the Collaboration. Be sure to choose the appropriate Collaboration Rule for each Collaboration.

   The properties and settings for these Collaborations are shown in **Figure 20 on page 68** through **Figure 22 on page 70**.
3. When you are finished, click **OK** to save your configuration and close the dialog box.

Figure 20 BabelFish Feeder Collaboration Properties

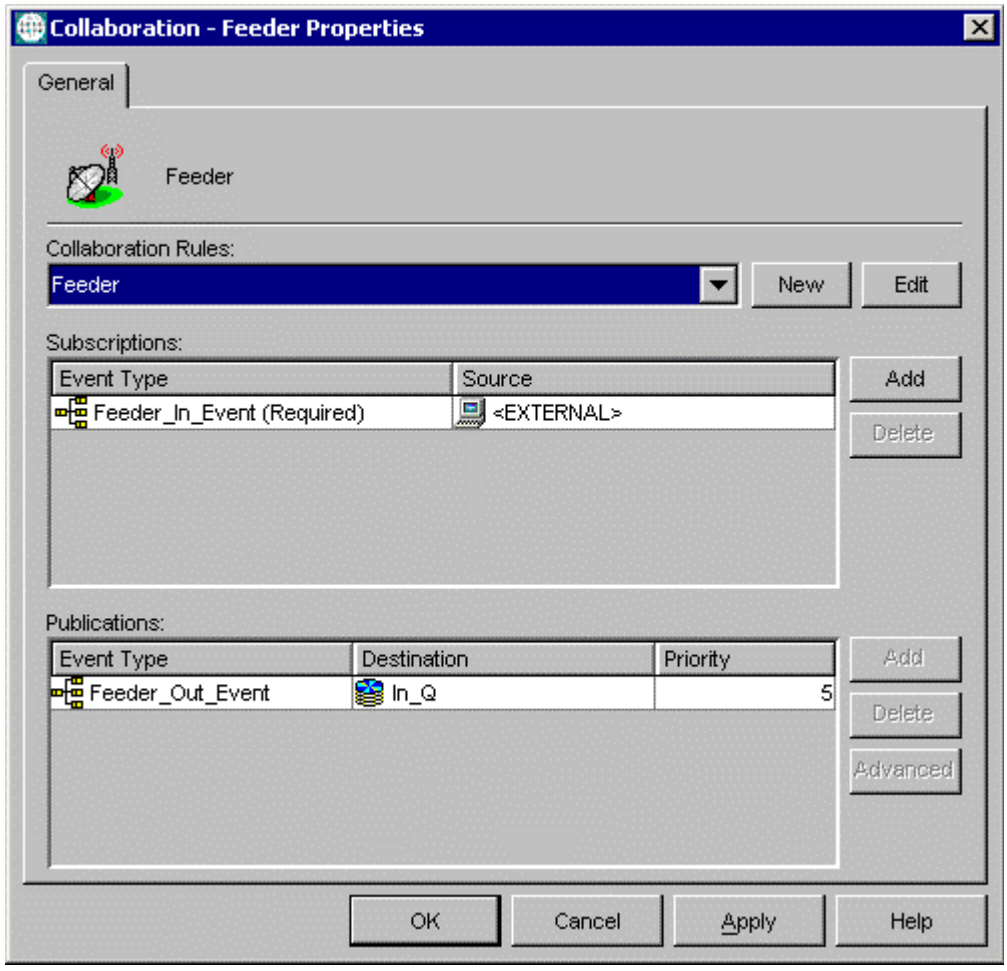


Figure 21 BabelFish Eater Collaboration Properties

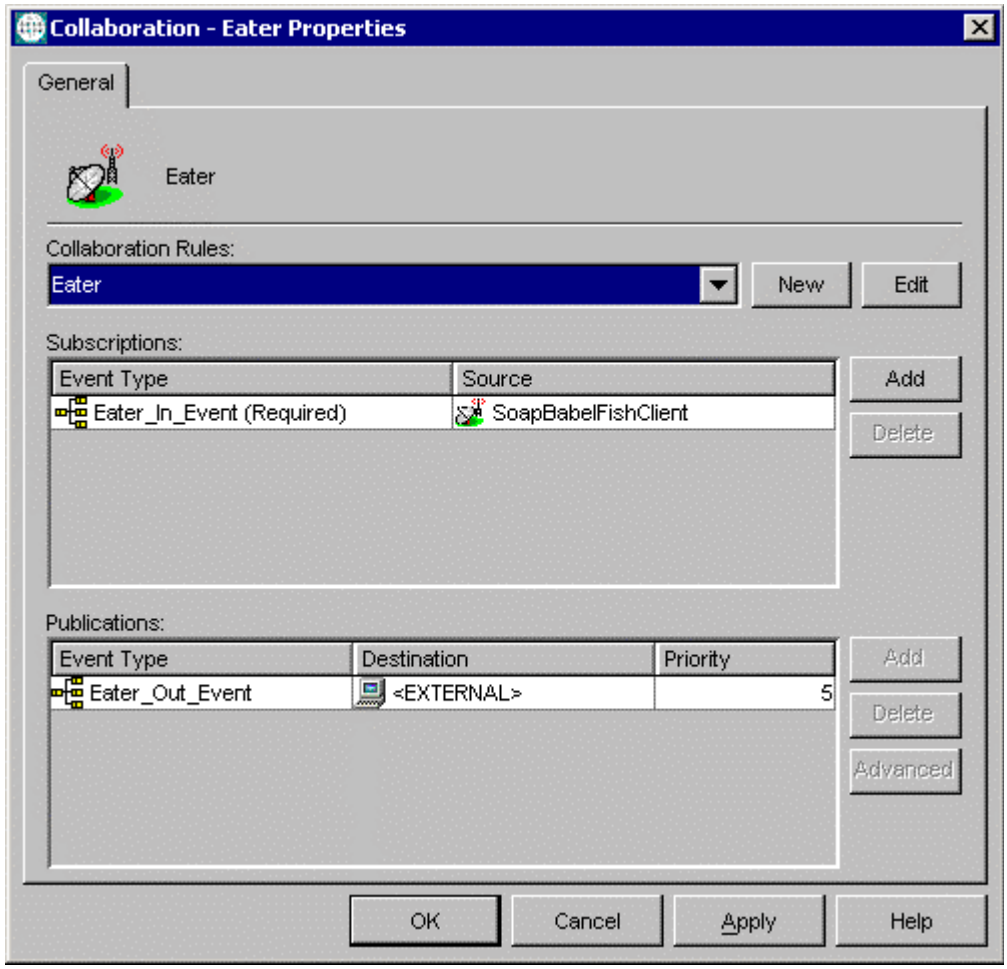
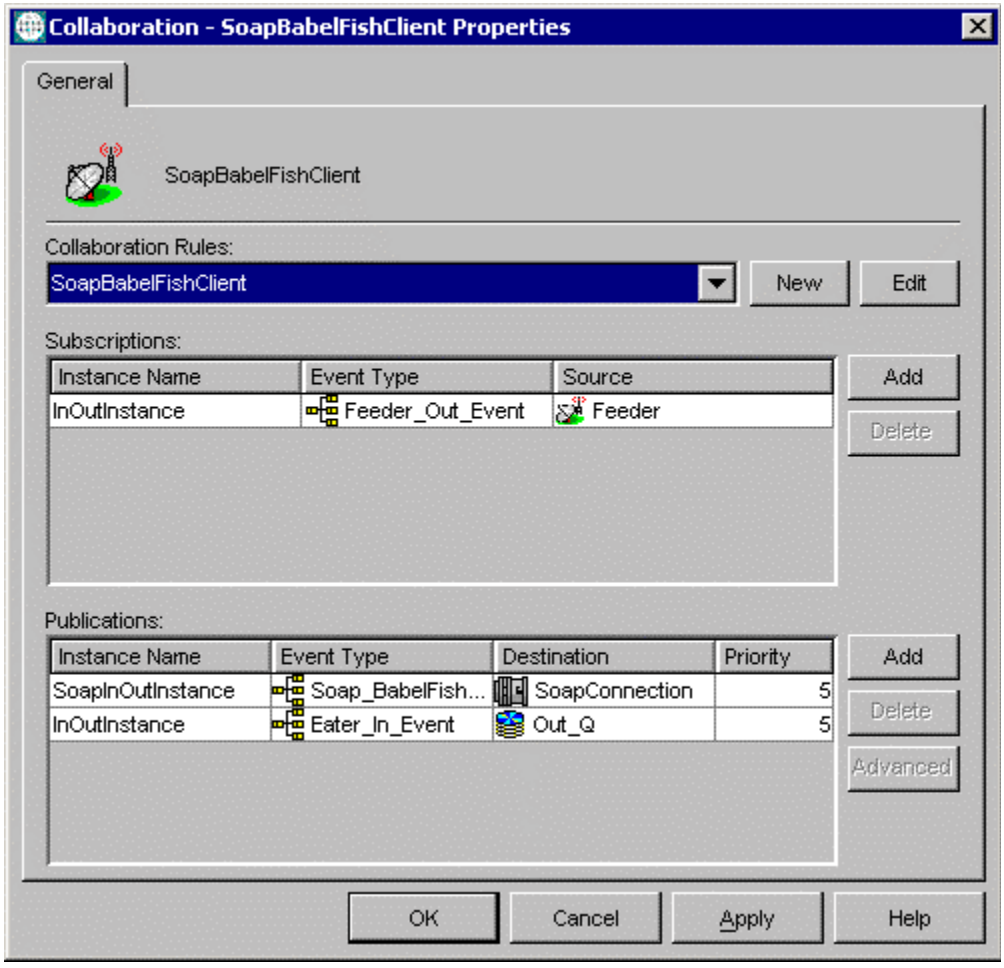


Figure 22 BabelFish SOAPBabelFishClient Collaboration Properties



Step 10: Test the Schema

Table 3 lists all the components for the schema. Check all the settings. Substitute the name of the machine running the schema for *host-name* where applicable.

Table 3 BabelFish Schema Components

Component	Logical Name	Settings
Schema	BabelFish	
Control Broker	*host-name*_cb	
IQ Manager	*host-name*_iqmgr	Start Up = Auto

Table 3 BabelFish Schema Components (Continued)

Component	Logical Name	Settings
Event Type	Feeder_In_Event	
	Feeder_Out_Event	
	Eater_In_Event	
	Eater_Out_Event	
	SOAP_BabelFish_Event	
Java ETD	BabelFish.xsc	▪ Package Name = pkgSOAPSample
Collaboration Rules	Feeder	▪ Service = Pass Through ▪ Subscription = Feeder_In_Event ▪ Publication = Feeder_Out_Event
	Eater	▪ Service = Pass Through ▪ Subscription = Eater_In_Event ▪ Publication = Eater_Out_Event
	SOAPBabelFishClient	▪ Service = Java ▪ Subscription = InOutInstance; Feeder_Out_Event (In; Trigger) ▪ Publication = SoapInOutInstance; SOAP_BabelFish_Event (Out) InOutInstance; Eater_In_Event (Out)
Java Collaboration Rule Class	JavaCollab.class	▪ Source = InOutInstance ▪ Destination = SoapInOutInstance
e*Way Connection	SoapConnection	▪ -1 for Event Type “get” interval
Inbound e*Way	Feeder	▪ Executable = stcewfile.exe ▪ Configuration file = Feeder.cfg ▪ Start Up = Auto ▪ Collaboration = Feeder
Outbound e*Way	Eater	▪ Executable = stcewfile.exe ▪ Configuration file = Eater.cfg ▪ Start Up = Auto ▪ Collaboration = Eater
Multi-Mode e*Way	SOAPBabelFishClient	▪ Executable = stceway.exe ▪ Configuration file = SOAPBabelFishClient.cfg ▪ Start Up = Auto ▪ Collaboration = SOAPBabelFishClient
IQ	In_Q	▪ Service = STC_Standard
	Out_Q	▪ Service = STC_Standard

Table 3 BabelFish Schema Components (Continued)

Component	Logical Name	Settings
Collaboration	Feeder	▪ Collaboration Rule = Feeder ▪ Subscription = Feeder_In_Event from <EXTERNAL> ▪ Publication = Feeder_Out_Event to In_Q
	Eater	▪ Collaboration Rule = Eater ▪ Subscription = Eater_In_Event from SOAPBabelFishClient ▪ Publication = Eater_Out_Event to <EXTERNAL>
	SOAPBabelFishClient	▪ Collaboration Rule = SOAPBabelFishClient ▪ Subscription = InOutInstance and Feeder_Out_Event from Feeder ▪ Publication = ♦ SoapInOutInstance and SOAP_BabelFish_Event to SoapConnection ♦ InOutInstance and Eater_In_Event to Out_Q

To run the BabelFish schema

1 From the command line prompt, enter:

```
stccb -rh hostname -rs schemaname -un username
-up user password -ln hostname_cb
```

Substitute ***hostname***, ***username***, ***schemaname***, and ***user password*** as appropriate.

2 Change the input file name extension to **.fin**.

The schema components start automatically. When there are no more run-time messages, check the output file. If the schema is operating correctly, you can see that this file contains the input text (good morning) translated into French (*bon jour*).

5.3 SOAP Receiver Implementation

This section explains how to implement a sample schema for the SOAP e*Way, for the SOAP receiver.

5.3.1 SOAP Receiver Schema: Overview

The SOAPClientAndServer schema demonstrates the use of the SOAP e*Way in implementing a Web server, a SOAP client, and a SOAP server.

Schema Operation

The sample schema contains elements that do the following operations:

- A pair of file e*Ways that communicate with the SOAP client and an external system via an IQ
- A SOAP client that posts data to and receives it from an ASP page on a Web server
- A JMS IQ Manager that exchanges data with a SOAP server and the Web server

The SOAP server implements an "add two numbers" service in an e*Gate Java Collaboration Rule within its Collaboration. All elements outlined in the previous paragraph, except the ASP page and external system, are within the e*Gate SOAPClientAndServer schema.

Schema Input Data

The text of the input data file is:

```
<number1>5</number1>
<number2>11</number2>
<sum>0</sum>
```

Schema Output Data

The **Feeder_eater** e*Way passes the input data to the **SOAPClient** e*Way which, in turn, sends the SOAP request to the **SOAPServiceImpl** e*Way (SOAP server) via the ASP page. The **JMS_CONN** e*Way Connection receives the SOAP response and passes it back to **Feeder_eater** via the Web server and **SOAPClient**. **Feeder_eater** then produces the output file.

The SOAP server adds the two numbers and returns the sum as follows:

```
<number1>5</number1>
<number2>11</number2>
<sum>16</sum>
```

Schema Components

This sample SOAPClientAndServer schema implementation consists of the following components:

e*Ways

- **Feeder_eater** file e*Way reads text from an external source, applies a Pass Through Collaboration Rule, and publishes the information to an IQ that stores inbound data. It also receives the outbound message from the same IQ and publishes it externally to a file.

- **SOAPClient** Multi-Mode e*Way applies extended Java Collaboration Rules to an inbound Event to translate the input data into SOAP and back again. In this case, the e*Way sends a SOAP message to a Web server, receives a processed text response from the SOAP service (via the Web server), and publishes the response to an IQ.
- **SOAPServiceImpl** Multi-Mode e*Way implements the SOAP service, also applying Java Collaboration Rules. It receives a SOAP message from the JMS IQ Manager and returns the processed Event to the SOAP client via the **JMS_CONN** e*Way Connection and Web server.

Event Types

- **GenericInEvent** contains raw data from the input file.
- **GenericOutEvent** contains the processed data output file.
- **TopicRequest** contains the known topic used by the ASP page to publish the SOAP message to the **JMS_CONN** e*Way Connection.

Collaboration Rules

- **feed** is associated with the **Feeder_eater** e*Way and is used for polling the input Event.
- **eat** is associated with the **Feeder_eater** e*Way and is used for sending the processed Event to the output file.
- **SOAPClient** is associated with the **SOAPClient** e*Way and is used to perform a transformation process (translating the input data into SOAP), send the SOAP request to the Web server, receive a SOAP response from the Web server, and publish that response to the **eat** Collaboration.
- **SOAPServiceImpl** is associated with the **SOAPServiceImpl** e*Way and is also used to do a transformation process, implementing the SOAP service. This Collaboration implements the SOAP service. **SOAPServiceImpl** receives the Event from the **JMS_CONN** e*Way Connection, does the calculation, and sends the Event back to **JMS_CONN**.

IQ

- **The_IQ** receives data from the **Feeder_eater** e*Way and sends it to the **SOAPClient** e*Way. It also sends and receives in the reverse direction.

Schema Configuration Notes

You must configure the **SOAPClient** e*Way and the JMS IQ Manager, then modify the ASP file so all of them can find each other. Ensure the following operations:

- The **SOAPClient** e*Way must publish to the URL for the ASP file.
- The ASP file must refer to the correct host name and port number of the JMS IQ Manager.

Location of Schema Files

The completed SOAPClientAndServer schema is included on the installation CD-ROM at the following location:

\samples\ewSOAP\SOAPServer.zip

To do this implementation, you first need to unzip the **SOAPServer.zip** file. The files listed in Table 4 are contained within this file.

Table 4 Sample Receiver Schema Files

File Name	Description
SOAPClientAndServer.zip	Export schema file
dtds\add2numsRequestBodyBlob.dtd	DTD that describes the BabelFish SOAP request; used to create the specific SOAP ETD for the sample
dtds\add2numsResponseBodyBlob.dtd	DTD that describes the BabelFish SOAP response; used to create the specific SOAP ETD for the sample
asp_jsp\TopicRequest.asp	ASP file to place on the Web server
asp_jsp\MS.inc	Additional file used by TopicRequest.asp to communicate with the JMS
data\input\data.fin	Input file
readme.html	Information file

To use this sample schema, the SOAP e*Way must be installed, the sample schema must be installed, and all of the necessary files and scripts must be located in the default location.

Schema Implementation

To implement this sample schema, you can do one of the following operations:

- To import the sample schema zip file, which automatically creates the sample schema components, see **"Sample Receiver Schema: Automatic Implementation" on page 75**.
- To manually create each of the components required to use the sample schema, see the instructions provided in **"Sample Receiver Schema: Manual Configuration" on page 77**.

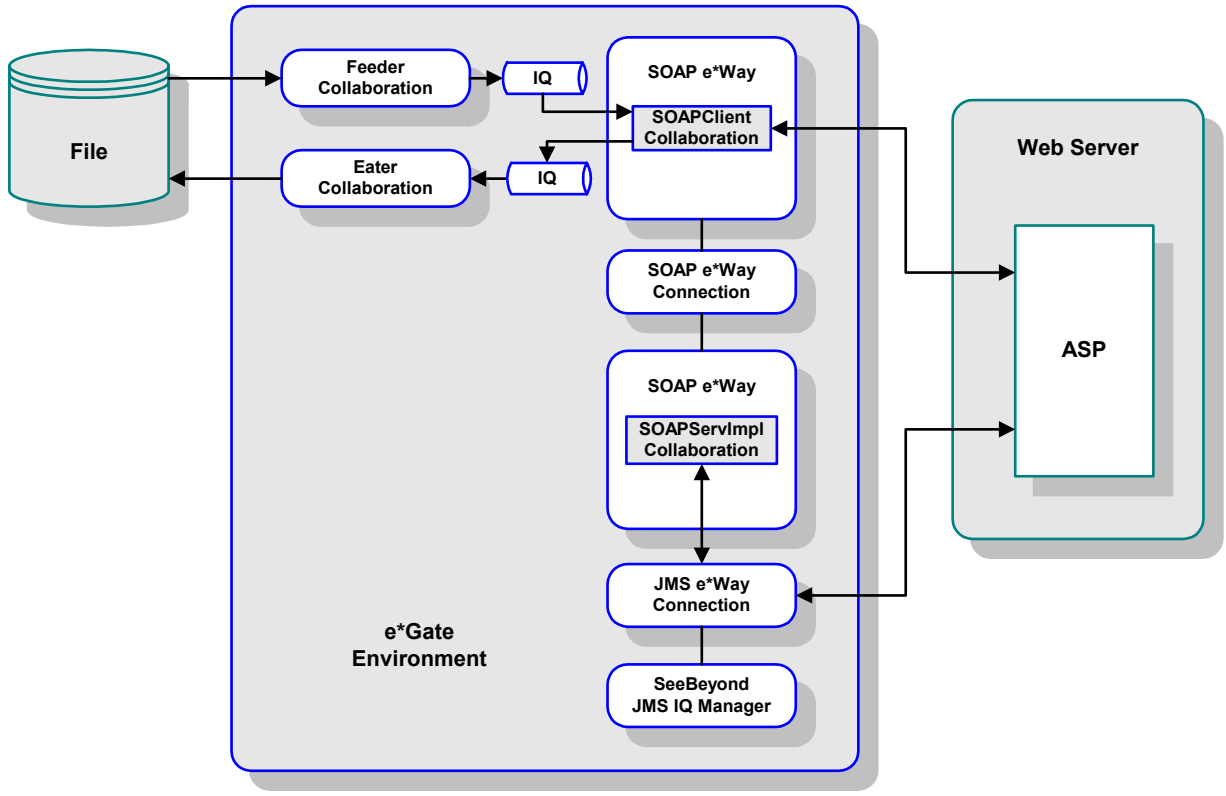
5.3.2 Sample Receiver Schema: Automatic Implementation

Install, configure, and run the SOAPClientAndServer sample schema in the same way as you did the BabelFish sample schema explained under **"Sample Sender Schema: Automatic Implementation" on page 42**.

The schema components start automatically. When there are no more run-time messages, check the output file. If the schema is operating correctly, this file contains the sum of the two numbers, 16.

Figure 23 on page 76 shows an overview diagram of the SOAPClientAndServer schema and how it operates.

Figure 23 SOAPClientAndServer Schema Overview



5.3.3 Sample Receiver Schema: Manual Configuration

This section describes how to configure the SOAPClientAndServer receiver schema manually in e*Gate.

Basic Implementation Steps

After you have located the SOAP service description, you must do the following steps:

1. Determine the SOAP endpoint URL on your Web server.
2. Determine the format of the SOAP message (see **"Step 2: Determine the Format of the SOAP Message" on page 46**).
3. Create a schema.
4. Create Event Types and Event Type Definitions (ETDs).
5. Create Collaboration Rules.
6. Create the e*Way Connection.
7. Add Intelligent Queues (IQs).
8. Add and configure e*Ways.
9. Create and configure Collaborations.
10. Test the schema.

See **"Sample Sender Schema: Manual Configuration" on page 44** for details on each of the previous steps.

Note: *For complete explanations of procedures in steps 3 through 10, see* ***Creating an End-to-End Scenario with e*Gate Integrator*** *and the* ***e*Gate Integrator User's Guide****.*

Table 5 lists all the components for the SOAPClientAndServer schema. Check all the settings. Substitute the name of the machine running the schema for *host-name* where applicable.

Table 5 SOAPClientAndServer Schema Components

Component	Logical Name	Settings
Schema	SOAPClientAndServer	
Control Broker	*host-name*_cb	
IQ Manager	localhost_iqmgr	Start Up = Auto
JMS IQ Manager	SBYN_JMS_QMGR	Start Up = Auto
Event Type	GenericInEvent	
	GenericOutEvent	
	TopicRequest	

Table 5 SOAPClientAndServer Schema Components (Continued)

Component	Logical Name	Settings
Java ETD	add2numbers.xsc	▪ Package Name = pkgSOAPSample
	TopicRequest.xsc	▪ Package Name = pkgSOAPSample
	SoapEvent.xsc	▪ Package Name = pkgSOAPSample
Collaboration Rules	feed	▪ Service = Pass Through ▪ Subscription = GenericInEvent ▪ Publication = GenericInEvent
	eat	▪ Service = Pass Through ▪ Subscription = GenericOutEvent ▪ Publication = GenericOutEvent
	SOAPClient	▪ Service = Java ▪ Subscription = in; GenericInEvent (In/Out; Trigger) ▪ Publication = soap; GenericInEvent; SOAP (In; Trigger) in; GenericOutEvent; The_Q (In/Out; Manual Publish)
	SOAPServiceImpl	▪ Service = Java ▪ Subscription = data; TopicRequest; JMS_CONN (In/Out; Trigger) ▪ Publication = soap; GenericInEvent; SOAP (In/Out; Manual Publish)
Java Collaboration Rule Class	SoapClient.class	▪ Source = in ▪ Destination = soap
	SOAPServiceImpl.class	▪ Source = data ▪ Destination = soap
e*Way Connection	SOAP	▪ 0 for Event Type “get” interval
	JMS_CONN	▪ 100 for Event Type “get” interval
Inbound/Outbound e*Way	Feeder_eater	▪ Executable = stcewfile.exe ▪ Configuration file = feeder_eater.cfg ▪ Start Up = Auto ▪ Collaborations = feed and eat
Multi-Mode e*Way	SOAPClient	▪ Executable = stceway.exe ▪ Configuration file = SOAPClient.cfg ▪ Start Up = Auto ▪ Collaboration = SOAPClient
	SOAPServiceImpl	▪ Executable = stceway.exe ▪ Configuration file = SOAPServiceImpl.cfg ▪ Start Up = Auto ▪ Collaboration = SOAPServiceImpl
IQ	The_Q	▪ Service = STC_Standard

Table 5 SOAPClientAndServer Schema Components (Continued)

Component	Logical Name	Settings
Collaboration	feed	▪ Collaboration Rule = feed ▪ Subscription = GenericInEvent from <EXTERNAL> ▪ Publication = GenericInEvent to The_Q
	eat	▪ Collaboration Rule = eat ▪ Subscription = GenericOutEvent from SOAPClient ▪ Publication = GenericOutEvent to <EXTERNAL>
	SOAPClient	▪ Collaboration Rule = SOAPClient ▪ Subscription = in and GenericInEvent from feed ▪ Publication = ♦ soap and GenericInEvent to SOAP ♦ in and GenericOutEvent to The_Q
	SOAPServiceImpl	▪ Collaboration Rule = SOAPServiceImpl ▪ Subscription = data and TopicRequest from JMS_CONN ▪ Publication = soap and GenericInEvent to SOAP

Contents of ASP File

The following text shows the contents of the ASP file:

```
<%@ Language=VBScript %>

<%
   'Ensure that this page is not cached.
   Response.Expires = 0
%>

<!--#include virtual ="ms.inc"-->
<%

   'Create the topic Connection Factory
   set topicConnectionFactory = server.CreateObject("STC_MSCOM.TopicC
onnectionFactory")

   'Set the hostname where the STC Message Server is running
   topicConnectionFactory.hostname = "xsongdell"

   'Set the port the STC Message Server is on
   topicConnectionFactory.port = "3600"

   'Create the topic Connection
   Set topicConnection = topicConnectionFactory.CreateTopicConnection
()

   'Create the topic Session
   Set topicSession = topicConnection.CreateTopicSession(false, msAut
oAcknowledge)
```

```
   'Start (or restart) a Connection's delivery of incoming messages.
Restart begins with the oldest unacknowledged message. Starting a sta
rted session is ignored.
   topicConnection.Start

   'Create the topic name to be used in the Message Server
   Set Topic = topicSession.CreateTopic("TopicRequest")

'---------------------------------------------------------------------
' The following sub routine reads the post in a binary append loop
'---------------------------------------------------------------------
      dim vntRequest()
      dim cRead, cOff, iAsc
      dim blnUseBinary, blnOK

      cRead = Request.TotalBytes

      ReDim vntRequest(cRead)

      strSend = ""

      do while cOff < cRead
         vntRequest(cOff) = Request.BinaryRead(1)
      strSend = strSend + Chr(AscB(vntRequest(cOff)))
      cOff = cOff + 1
      loop

      blnOK = true

'---------------------------------------------------------------------

   'Create the message that is to be published and assigns it the con
tents of variable strSend, the ""+ is there to force string type
   set MessagePublished = topicSession.CreateTextMessage(""+strSend)

   'Creates the Topic Requestor object
   set TopicRequestor = server.CreateObject("STC_MSCOM.TopicRequestor
")

   'Initialize the Topic Requestor
   TopicRequestor.Create topicSession, topic

   'This line will send the message set in MessagePublished and recei
ve the unique reply into MessageReceived
   set MessageReceived = topicRequestor.request(MessagePublished)

   'This is an asp command to write the received response to the scre
en
   Response.Write(MessageReceived.Text)

%>
```

Running the SOAPClientAndServer Schema

For details on how to run the SOAPClientAndServer schema, see **"To run the BabelFish schema" on page 72**. The schema components start automatically. When there are no more run-time messages, check the output file. If the schema is operating correctly, you can see that this file contains the sum of the two numbers, 16, in the last line.

Chapter 6

Java Methods

This chapter explains the Java methods used in the e*Way Intelligent Adapter for SOAP.

6.1 SOAP e*Way Methods and Classes: Overview

For any e*Way, communication takes place both on the e*Gate system and the external system side. Communication between the e*Way and the e*Gate environment is common to all e*Ways, while the communication between the e*Way the external system is different for each e*Way.

For the SOAP e*Way, the **stceway.exe** file is used to communicate between the e*Way and e*Gate, and a Java Collaboration is utilized to keep the communication open between the e*Way and the SOAP service.

Java methods have been added to make it easier to set information in the **SOAPEvent** Event Type Definition (ETD) and get information from it. The nature of this data transfer depends on the Document Type Definition (DTD) processed by the SOAP wizard in the e*Gate Enterprise Manager.

The Java methods for the SOAP e*Way are contained in the following classes:

- "SOAPTransport Class" on page 124
- "SOAPVerification Class" on page 127

6.2 Attribute Class

The **Attribute** class represents an attribute of a SOAP ETD and is used to generate attributes of a SOAP header, SOAP body, or SOAP envelope node.

The **Attribute** class is defined as:

```
public class Attribute
```

The **Attribute** class extends **java.lang.Object.**

The **Attribute** class methods include:

- **getKey** on page 82
- **getValue** on page 82
- **setKey** on page 83
- **setValue** on page 83

getKey

Description

getKey gets the key to this attribute.

Syntax

```
public java.lang.String getKey()
```

Parameters

None.

Returns

java.lang.String
The key to this attribute; can be null.

Throws

None.

getValue

Description

getValue retrieves the value of this attribute.

Syntax

```
public java.lang.String getValue()
```

Parameters

None.

Returns

java.lang.String
The value of this attribute; can be null.

Throws

None.

setKey

Description

setKey sets the key to this attribute. **_key** must be a qualified name.

Syntax

```
public void setKey(java.lang.String _key)
```

Parameters

Name	Type	Description
_key	String	The key to this attribute; can be null.

Returns

Void.

Throws

java.lang.Exception when any generic error occurs.

setValue

Description

setValue sets the value to this attribute.

Syntax

```
public void setValue(java.lang.String _value)
```

Parameters

Name	Type	Description
_value	String	The key to this attribute; can be null.

Returns

Void.

Throws

None.

6.3 SOAP Class

Implementation of a SOAP ETD. This ETD allows for processing of SOAP messages. The SOAP ETD unmarshals valid Extensible Markup Language (XML) documents to its **SOAPRequest** node. A typical client use of this ETD would be to unmarshal a well-formed SOAP message (to the **SOAPRequest** node), call the **sendToSOAPServer** method, which unmarshals the response into the **SOAPResponse** node.

Another client use of the ETD would be to fill in key fields of the **SOAPRequest** node, then call the **sendToSOAPServer** method, which unmarshals the response into the **SOAPResponse** node.

A SOAP ETD can also be used to unmarshal a SOAP request and form a SOAP response, enabling implementation of SOAP services within a Collaboration.

The **SOAP** class is defined as:

```
public class SOAP
```

The **SOAP** class extends **com.stc.jcsre.MsgETDImpl.**

The **SOAP** class methods include:

getSOAPActionURI

Description

getSOAPActionURI gets the **SOAPAction** header used to send the request message.

Syntax

```
public java.lang.String getSOAPActionURI()
```

Parameters

None.

Returns

java.lang.String
The **SOAPAction** URI.

Throws

None.

getSOAPRequest

Description

getSOAPRequest gets the **SOAPRequest** object.

Syntax

```
public SOAPRequest getSOAPRequest()
```

Parameters

None.

Returns

Object
The **SOAPRequest** object.

Throws

None.

getSOAPResponse

Description

getSOAPResponse retrieves the **SOAPResponse** object.

Syntax

```
public SOAPResponse getSOAPResponse()
```

Parameters

None.

Returns

Object
The **SOAPResponse** object.

Throws

None.

getSOAPTransport

Description

getSOAPTransport retrieves the SOAPTransport object.

Syntax

```
public SOAPResponse getSOAPTransport()
```

Parameters

None.

Returns

Object
The **SOAPTransport** object.

Throws

None.

getURL

Description

getURL retrieves the URL to which the request message is sent.

Syntax

```
public java.lang.String getURL()
```

Parameters

None.

Returns

java.lang.String
The URL to which the request message is sent.

Throws

None.

marshal

Description

marshal marshals the data content of the ETD into a byte array. The default behavior is to marshal the **SOAPRequest** node. It overrides **marshal** in the class **com.stc.jcsre.SimpleETDImpl**.

Syntax

```
public byte[] marshal()
```

Parameters

None.

Returns

byte[]

Byte array of the BLOB result from marshaling.

Throws

com.stc.jcsre.MarshalException when it is unable to marshal ETD.

marshalRequest

Description

marshalRequest provides a convenient method to marshal the **SOAPRequest** node into a byte array.

Syntax

```
public byte[] marshalRequest()
```

Parameters

None.

Returns

byte[]

The byte array element representing the request.

Throws

com.stc.jcsre.MarshalException if it is unable to marshal the **SOAPRequest** node.

marshalResponse

Description

marshalResponse provides a convenient method to marshal the **SOAPResponse** node into an XML document, returning it as a byte array.

Syntax

```
public byte[] marshalResponse()
```

Parameters

None.

Returns

byte[]

The response is a valid SOAP XML document converted into a byte array.

Throws

com.stc.jcsre.MarshalException if it is unable to marshal the **SOAPResponse** node.

receiveRequest

Description

receiveRequest gets a message from the IQ for a specific topic name and unmarshals it into the **SOAPRequest** node.

Syntax

```
public boolean receiveRequest(java.lang.String _topicName)
```

Parameters

Name	Type	Description
_topicName	String	The name of the topic to subscribe to.

Returns

Boolean

True if it is able to receive and unmarshal the SOAP message into the **SOAPRequest** node.

Throws

com.stc.common.collabService.CollabDataException if any error occurs.

receiveResponse

Description

receiveResponse gets a message from the IQ for a specific topic name and unmarshal it into the **SOAPResponse** node.

Syntax

```
public boolean receiveResponse()
```

Parameters

Name	Type	Description
`_topicName`	String	The name of the topic to subscribe to.

Returns

Boolean

True if able to receive and unmarshal the SOAP message from the given topic into the **SOAPRequest** node.

Throws

`Throws com.stc.common.collabService.CollabDataException` if any error occurs.

reset

Description

reset resets the data content of an ETD. It overrides **reset** in the class **com.stc.jcsre.SimpleETDImpl**.

Syntax

```
public boolean reset()
```

Parameters

None.

Returns

Boolean

Returns **true** if the reset clears data content of the ETD, **false** if the ETD does not have meaningful implementation of **reset()**, thus necessitating the creation of a new ETD.

sendRequest

Description

sendRequest marshals the **SOAPRequest** node and puts a message to the Intelligent Queue (IQ) with the given topic name.

Syntax

```
public void sendRequest(java.lang.String _topicName)
```

Parameters

Name	Type	Description
_topicName	String	The name of the topic to send the request to.

Returns

Void.

Throws

com.stc.common.collabService.CollabDataException if any error occurs.

sendResponse

Description

sendResponse marshals the **SOAPResponse** node and puts the message to the IQ.

Syntax

```
public Void. sendResponse()
```

Parameters

None.

Throws

com.stc.common.collabService.CollabDataException if any error occurs.

setSOAPActionURI

Description

setSOAPActionURI sets the **SOAPAction** URI used in sending the request message.

Syntax

```
public void setSOAPActionURI(java.lang.String _soapActionURI)
```

Parameters

Name	Type	Description
_soapActionURI	String	The SOAPActionURI to set.

Returns

Void.

Throws

None.

setSOAPRequest

Description

setSOAPRequest sets the **SOAPRequest** object.

Syntax

```
public void setSOAPRequest(SOAPRequest _request)
```

Parameters

Name	Type	Description
_request	Object	The SOAPRequest object to set.

Returns

Void.

Throws

None.

setSOAPResponse

Description

setSOAPResponse sets the **SOAPResponse** object.

Syntax

```
public void setSOAPResponse(SOAPResponse _response)
```

Parameters

Name	Type	Description
_response	Object	The SOAPResponse object to set.

Return Values

Void.

Throws

None.

setSOAPTransport

Description

setSOAPTransport sets the **SOAPTransport** object.

Syntax

```
public void setSOAPTransport(SOAPResponse _response)
```

Parameters

Name	Type	Description
_response	Object	The SOAPResponse object to set.

Parameters

None.

Returns

Void.

Throws

None.

setURL

Description

setURL sets the URL to which the request message is sent.

Syntax

```
public java.lang.String setURL(java.lang.String _url)
```

Parameters

Name	Type	Description
_url	String	The URL to set.

Returns

java.lang.String
The URL to which the request message is sent.

Throws

java.net.MalformedURLException when **_url** is an invalid URL.

unmarshal

Description

unmarshal unmarshals a byte array into the data content of an ETD. The default behavior is to unmarshal to the **SOAPRequest** node. It overrides **unmarshal** in the class **com.stc.jcsre.SimpleETDImpl**.

Syntax

```
public void unmarshal(byte[] _blob)
```

Parameters

Name	Type	Description
_blob	Byte array	Byte array of the XML document to be unmarshaled.

Returns

Void.

Throws

com.stc.jcsre.UnmarshalException when it is unable to unmarshal **_blob** into the ETD.

unmarshalRequest

Description

unmarshalRequest provides a convenient method to unmarshal a byte array into the **SOAPRequest** node.

Syntax

```
public void unmarshalRequest(byte[] _blob)
```

Parameters

Name	Type	Description
byte[]	Byte array	The XML document to unmarshal.

Returns

Void.

Throws

None.

unmarshalResponse

Description

unmarshalResponse provides a convenient method to unmarshal a byte array into the **SOAPResponse** node. The byte array must be a valid XML document.

Syntax

```
public void unmarshalResponse(byte[] _blob)
```

Parameters

Name	Type	Description
byte[]	XML	The XML document to unmarshal, represented as a byte array.

Returns

Void.

Throws

com.stc.jcsre.UnmarshalException if it is unable to unmarshal the **SOAPResponse** node.

6.4 SOAPAttachment Class

The **SOAPAttachment** class is an ETD node that represents an attachment to a SOAP message.

The **SOAPAttachment** class is defined as:

```
public class SOAPAttachment
```

The **SOAPAttachment** class extends **java.lang.Object.**

The **SOAPAttachment** class methods include:

- **setName** on page 99
- **setTransferEncoding** on page 99
- **setValue** on page 100

addReference

Description

addReference adds a reference to **_obj**. The parameter **_obj** must implement the ETD interface and have the **setHref** method. In other words, the XML element that the ETD represents must have an **Href** attribute.

Syntax

```
public void addReference(com.stc.jcsre.ETD _obj)
```

Parameters

Name	Type	Description
_obj	Object	Object implementing the ETD interface.

Returns

Void.

Throws

- **java.lang.NoSuchMethodException** if a **setHref** method does not exist
- **java.lang.IllegalAccessException** if the **setHref** method does not have sufficient access privileges.
- **java.lang.reflect.InvocationTargetException**

base64Encode

Description

base64Encode encodes the current attachment in base 64. By default, the **Content-Transfer-Encoding** is set to **base64**.

Syntax

```
public void base64Encode()
```

Parameters

None.

Returns

Void.

Throws

java.io.IOException if it is unable to encode the contents of this attachment in base 64**.**

getContentType

Description

getContentType retrieves the **Content-Type** for the current attachment. By default, the content type of the attachment is set to **application/octet-stream** if no content type is specified.

Syntax

```
public java.lang.String getContentType()
```

Parameters

None.

Returns

java.lang.String
The content type for the current attachment.

Throws

None.

getFileLocation

Description

getFileLocation retrieves the file name of the current attachment.

Syntax

```
public java.lang.String getFileLocation()
```

Parameters

None.

Returns

java.lang.String
The file name of the current attachment.

Throws

None.

getName

Description

getName retrieves the name of the current attachment. This name is used to generate the Content-ID and must be unique across all attachments.

Syntax

```
public java.lang.String getName()
```

Parameters

None.

Returns

java.lang.String
The attachment name.

Throws

None.

getTransferEncoding

Description

getTransferEncoding retrieves the content transfer encoding for the current attachment. By default, **Content-Transfer-Encoding** is set to **binary** if no other **Content-Transfer-Encoding** is set.

Syntax

```
public java.lang.String getTransferEncoding()
```

Parameters

None.

Returns

java.lang.String
The content transfer encoding for the current attachment.

Throws

None.

getValue

Description

getValue retrieves the value of the attachment. If the value was set previously, it returns the value. If the value was not set, but a file location was specified, it retrieves the contents of the file and returns those contents as a byte array.

Syntax

```
public byte[] getValue()
```

Parameters

None.

Returns

byte[]
The value of the current attachment or the file contents as a byte array.

Throws

None.

setContentType

Description

setContentType sets the content type for the current attachment. The possible values are defined by RFC.

Syntax

```
public void setContentType(java.lang.String _contentType)
```

Parameters

Name	Type	Description
_contentType	String	The content type for this attachment.

Returns

Void.

Throws

None.

setFileLocation

Description

setFileLocation sets the file name of the current attachment. This method checks the following properties before setting the file name:

- The file must exist.
- The file must be a valid file and not a directory or a symbolic link.
- The file must be readable.

This method clears out any previous setting of the value field.

Syntax

```
public void setFileLocation(java.lang.String _fileLocation)
```

Parameters

Name	Type	Description
_fileLocation	String	The name of the file.

Returns

Void.

Throws

None.

setName

Description

setName sets the name of the current attachment. This name is used to generate the Content-ID and must be unique across all attachments.

Syntax

```
public void setName(java.lang.String _name)
```

Parameters

Name	Type	Description
_name	String	The unique name for this attachment; it cannot be null.

Returns

Void.

Throws

NullPointerException if **_name** is null.

setTransferEncoding

Description

setTransferEncoding sets the content transfer encoding to **_transferEncoding**. The possible values are defined by RFC.

Syntax

```
public void setTransferEncoding(java.lang.String _transferEncoding)
```

Parameters

Name	Type	Description
_transferEncoding	String	The content transfer encoding for this attachment.

Returns

Void.

Throws

None.

setValue

Description

setValue sets the value of the current attachment. If a file location was previously specified, it is set to the empty string.

Syntax

```
public void setValue(byte[] _value)
```

Parameters

Name	Type	Description
_value	Byte array	The value to set.

Returns

Void.

Throws

None.

6.5 SOAPBody Class

The **SOAPBody** class represents the **SOAPBody** element of the SOAP ETD, and is used to generate a body element in a SOAP message.

The **SOAPBody** class is defined as:

```
public class SOAPBody
```

The **SOAPBody** class extends **java.lang.Object.**

The **SOAPBody** class methods include:

getAttribute

Description

getAttribute allows a user to access a specific attribute.

Syntax

```
public Attribute getAttribute(int _index)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.

Returns

Attribute
The key-value pair that is the attribute.

Throws

None.

getBodyContents

Description

getBodyContents gets the contents of the specified **SOAPBody** object.

Syntax

```
public java.lang.String getBodyContents()
```

Parameters

None.

Returns

java.lang.String
A string representing the body contents of the **SOAPBody**. The contents must be well-formed XML.

Throws

None.

getNumberOfAttributes

Description

getNumberOfAttributes returns the number of attributes for this element.

Syntax

```
public int getNumberOfAttributes()
```

Parameters

None

Returns

Integer
The number of attributes for this element.

Throws

None.

getSOAPFault

Description

getSOAPFault retrieves the SOAP fault if one is generated.

Syntax

```
public SOAPFault getSOAPFault()
```

Parameters

None.

Returns

Object
The **SOAPFault** object; can be null if no fault was set.

Throws

None.

setAttribute

setAttribute allows you to set the index to a specific attribute.

Syntax

```
public void setAttribute(int _index, Attribute _attribute)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.
_attribute	Attribute	The key-value pair mapping.

Returns

Void.

Throws

None.

setBodyContents

Description

setBodyContents sets the contents of the **SOAPBody** object to the given string.

Syntax

```
public void setBodyContents(java.lang.String _bodyContents)
```

Parameters

Name	Type	Description
_bodyContents	String	A string representing the contents of the body of a SOAP message; the message must be well-formed XML.

Returns

Void.

Throws

java.lang.Exception when any generic error occurs.

setSOAPFault

Description

setSOAPFault sets the SOAP fault to the specified object.

Syntax

```
public void setSOAPFault(SOAPFault _soapFault)
```

Parameters

Name	Type	Description
_soapFault	Object	An object representing a SOAP fault.

Returns

Void.

Throws

None.

6.6 SOAPFault Class

The **SOAPFault** class represents the **SOAPFault** node in the SOAP ETD and is used to generate fault elements of a SOAP message.

The **SOAPFault** class is defined as:

```
public class SOAPFault
```

The **SOAPFault** class extends **java.lang.Object.**

The **SOAPFault** class methods include:

getDetail

Description

getDetail gets the detail information for this fault if it exists.

Syntax

```
public java.lang.String getDetail()
```

Parameters

None.

Returns

java.lang.String
The detail string.

Throws

None.

getFaultActor

Description

getFaultActor gets the fault actor URI.

Syntax

```
public java.lang.String getFaultActor()
```

Parameters

None.

Returns

java.lang.String
The fault actor URI.

Throws

None.

getFaultCode

Description

getFaultcode gets the fault code.

Syntax

```
public java.lang.String getFaultCode()
```

Parameters

None.

Returns

java.lang.String
The fault code.

Throws

None.

getFaultString

Description

getFaultString gets the fault string.

Syntax

```
public java.lang.String getFaultString()
```

Parameters

None.

Returns

java.lang.String
The fault string.

Throws

None.

setDetail

Description

getDetail sets the detail information for this fault if it exists.

Syntax

```
public void getDetail(java.lang.String _detail)
```

Parameters

Name	Type	Description
_detail	String	The detail information as well-formed XML.

Returns

Void.

Throws

java.lang.Exception when **_detail** is not well-formed XML.

setFaultActor

Description

setFaultActor sets the fault actor.

Syntax

```
public void setFaultActor(java.lang.String _actor)
```

Parameters

Name	Type	Description
_actor	String	The fault actor.

Returns

Void.

Throws

None.

setFaultCode

Description

setFaultCode sets the fault code.

Syntax

```
public void SetFaultCode(java.lang.String _code)
```

Parameters

Name	Type	Description
_code	String	The fault code.

Returns

Void.

Throws

None.

setFaultString

Description

setFaultString sets the fault string.

Syntax

```
public void setFaultString(java.lang.String _string)
```

Parameters

Name	Type	Description
_string	String	The fault string.

Returns

Void.

Throws

None.

6.7 SOAPHeader Class

The **SOAPHeader** class directly represents the **SOAPHeader** node of the SOAP ETD, and is used to generate the header element of the SOAP message.

The **SOAPHeader** class is defined as:

```
public class SOAPHeader
```

The **SOAPHeader** class extends **java.lang.Object.**

The **SOAPHeader** class methods include:

getAttribute

Description

getAttribute allows a user to access a specific attribute.

Syntax

```
public Attribute getAttribute(int _index)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.

Returns

Attribute
The key-value pair that is the attribute.

Throws

None.

getHeaderContents

Description

getHeaderContents gets the contents of the **SOAPHeader** node to the specified string.

Syntax

```
public java.lang.String getHeaderContents()
```

Parameters

None.

Returns

java.lang.String
The contents of the header as a string. The contents must be well-formed XML.

Throws

None.

getNumberOfAttributes

Description

getNumberOfAttributes returns the number of attributes for this element.

Syntax

```
public int getNumberOfAttributes()
```

Parameters

None

Returns

Integer
The number of attributes for this element.

Throws

None.

setAttribute

Description

setAttribute allows a user to set the index to a specific attribute.

Syntax

```
public void setAttribute(int _index, Attribute _attribute)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.
_attribute	Attribute	The key-value pair mapping.

Returns

Void.

Throws

None.

setHeaderContents

Description

setHeaderContents sets the contents of the **SOAPHeader** node. The contents must be well-formed XML.

Syntax

```
public void setHeaderContents(java.lang.String _headerContents)
```

Parameters

Name	Type	Description
_headerContents	String	The contents of the header.

Returns

Void.

Throws

java.lang.Exception when any content error occurs.

6.8 SOAPMessage Class

The **SOAPMessage** class serves as a base class for all ETD nodes that are SOAP messages. This class is used to generate a complete SOAP envelope with required body and optional header and fault elements.

The **SOAPMessage** class is defined as:

```
public class SOAPMessage
```

The **SOAPMessage** class extends **java.lang.Object.**

The **SOAPMessage** class methods include:

getAttribute

Description

getAttribute allows you to access a specific attribute.

Syntax

```
public Attribute getAttribute(int _index)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.

Returns

Attribute
The key-value pair that makes up the attribute.

Throws

None.

getNumberOfAttributes

Description

getNumberOfAttributes returns the number of attributes for this element.

Syntax

```
public int getNumberOfAttributes()
```

Parameters

None

Returns

Integer
The number of attributes for this element.

Throws

None.

getSOAPBody

Description

getSOAPBody returns an ETD-specific **SOAPBody** object.

Syntax

```
public SOAPBody getSOAPBody()
```

Parameters

None.

Returns

Object
The body of this **SOAPMessage**.

Throws

None.

getSOAPHeader

Description

getSOAPHeader returns an ETD-specific **SOAPHeader** object.

Syntax

```
public SOAPHeader getSOAPHeader()
```

Parameters

None.

Returns

Object
The header of the **SOAPMessage.**

Throws

java.lang.Exception when any generic error occurs.

marshal

Description

marshal marshals the contents of the **SOAPMessage** class into a byte array.

Syntax

```
public byte[] marshal()
```

Parameters

None.

Returns

byte[]
Byte array representation of this message.

Throws

com.stc.jcsre.MarshalException if it is unable to marshal the contents of this object into a byte array.

setAttribute

Description

setAttribute allows you to set the index to a specific attribute.

Syntax

```
public void setAttribute(int _index, Attribute _attribute)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.
_attribute	Attribute	The key-value pair mapping.

Returns

Void.

Throws

None.

setSOAPBody

Description

setSOAPBody sets the ETD-specific SOAP body.

Syntax

```
public void setSOAPBody(SOAPBody _soapbody)
```

Parameters

Name	Type	Description
_soapBody	SOAPBody	A new SOAPBody.

Returns

Void.

Throws

None.

setSOAPHeader

Description

setSOAPHeader sets the ETD-specific SOAP header.

Syntax

```
public void setSOAPHeader(SOAPHeader _soapHeader)
```

Parameters

Name	Type	Description
_soapHeader	SOAPHeader	A new SOAP header.

Returns

Void.

Throws

None.

unmarshal

Description

unmarshal unmarshals the byte array **_blob** into the internal structure of this **SOAPMessage** class. It is assumed that **_blob** is a valid, well-formed XML document conforming to the SOAP specification.

Syntax

```
public void unmarshal(byte[] _blob)
```

Parameters

Name	Type	Description
_blob	Byte array	The byte array representing a SOAP message.
SOAPAction URI	String	The identifier from which the server is told what action to take.
Transport Type	String	The transport mechanism, for example, HTTP, SMTP, or HTTP(S). Currently, HTTP is the only transport mechanism supported.

Returns

Void.

Throws

com.stc.jcsre.UnmarshalException if it is unable to interpret **_blob** into the internal class attributes.

6.9 SOAPNode Class

The **SOAPNode** class is a base class for all SOAP-specific nodes in an ETD. This class implements the ETD interface and extends **XMLETDImpl**. This class represents a generic, SOAP element from a SOAP XML document. For example, the **SOAPBody** class that extends from **SOAPNode** represents the element **SOAP-ENV:Body**.

SOAPNode is responsible for dealing with attributes and namespaces.

The **SOAPNode** class is defined as:

```
public class SOAPNode
```

The **SOAPNode** class extends **com.stc.jcsre.XMLETDImpl.**

The **SOAPNode** class methods include:

countAttribute

Description

countAttribute retrieves the number of attributes for this element.

Syntax

```
public int countAttribute()
```

Parameters

None.

Returns

Integer
The number of attributes for this element.

Throws

None.

getAttribute

Description

getAttribute allows a user to access a specific attribute. If no attribute exists at the given location, an empty attribute is created and returned.

Syntax

```
public Attribute getAttribute(int _index)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.

Returns

Attribute
The key-value pair that is the attribute.

Throws

None.

getLocalName

Description

getLocalName retrieves the local name associated with the current XML element.

Note: *Every SOAPNode class in a SOAP ETD represents an XML element.*

Syntax

```
public abstract java.lang.String getLocalName()
```

Parameters

None.

Returns

java.lang.String
The local name of the current ETD node.

Throws

None.

setAttribute

Description

setAttribute allows you to set the index to a specific attribute.

Syntax

```
public void setAttribute(int _index,Attribute _attribute)
```

Parameters

Name	Type	Description
_index	Integer	The location of the attribute.
_attribute	Attribute	The key-value pair mapping.

Returns

Void.

Throws

None.

unmarshal

Description

unmarshal unmarshals the byte array **_blob** into the current object. It overrides **unmarshal** in the class **com.stc.jcsre.XMLETDImpl**.

Syntax

```
public void unmarshal(byte[] _blob)
```

Parameters

Name	Type	Description
_blob	Byte array	The byte array to unmarshal.

Returns

Void.

Throws

com.stc.jcsre.UnmarshalException if an error occurs in interpreting the bytes.

6.10 SOAPRequest Class

The **SOAPRequest** class represents a **SOAPMessage** that is sent to a SOAP Server. Fundamentally, a **SOAPRequest** is a simple wrapper around a **SOAPMessage** object. This class represents the SOAP request of a SOAP ETD.

The **SOAPRequest** class is defined as:

```
public class SOAPRequest
```

The **SOAPRequest** class extends **SOAPMessage.**

The following **SOAPRequest** class methods were inherited from the **SOAPMessage** class, and are described under **SOAPMessage Class** on page 110:

- **getAttribute** on page 111
- **getSOAPBody** on page 112
- **getSOAPHeader** on page 112
- **marshal** on page 113
- **setAttribute** on page 113
- **setSOAPBody** on page 113
- **setSOAPHeader** on page 114
- **unmarshal** on page 114

6.11 SOAPResponse Class

The **SOAPResponse** class represents a **SOAPMessage** that has been received from a SOAP server. A SOAP response has the responsibility of dealing with possible SOAP exceptions and faults in addition to its regular responsibilities as a **SOAPMessage**.

The **SOAPResponse** class is defined as:

```
public class SOAPResponse
```

The **SOAPResponse** class extends **SOAPMessage.**

The following **SOAPResponse** class methods were inherited from the **SOAPMessage** class, and are described under that section (**SOAPMessage Class** on page 110):

- **getAttribute** on page 111
- **getSOAPBody** on page 112
- **getSOAPHeader** on page 112
- **marshal** on page 113
- **setAttribute** on page 113
- **setSOAPBody** on page 113
- **setSOAPHeader** on page 114
- **unmarshal** on page 119

The rest of this section explains the **SOAPResponse** class method not inherited from the **SOAPMessage** class.

unmarshal

Description

unmarshal behaves like a normal SOAP message, but be sure to check whether there are any faults. If any fault exists, it marshals that data to the fault node. It overrides **unmarshal** in the class **SOAPMessage**.

Syntax

```
public void unmarshal(byte[] blob)
```

Parameters

Name	Type	Description
blob	Byte array	The array that holds the XML document.

Returns

Void.

Throws

com.stc.jcsre.UnmarshalException if there is a fault.

6.12 SOAPSignature Class

The **SOAPSignature** class represents a **SOAP-SEC:Signature** element. This element encapsulates an XML signature element and is part of the header of the SOAP envelope.

Users of the SOAP ETD do not have to interact directly with this class. Through the use of the sign-and-verify method calls in **SOAPSigner** and **SOAPVerification**, respectively, objects of this class are generated automatically.

The **SOAPSignature** class is defined as:

```
public class SOAPSignature
```

The **SOAPSignature** class extends **SOAPNode.**

The **SOAPSignature** class methods include:

getLocalName

Description

getLocalName allows you to retrieve the local name of the current EDT node. It overrides **getLocalName** in the class **SOAPNode** (see **"SOAPNode Class" on page 115**).

Syntax

```
public java.lang.String getLocalName()
```

Parameters

None.

Returns

java.lang.String
The local name of the current node.

Throws

None.

getXMLSignature

Description

getXMLSignature retrieves the XML signature as a string. The XML signature must be well-formed XML and conform to the official XML Digital Signature Specification.

Syntax

```
public java.lang.String getXMLSignature()
```

Parameters

None.

Returns

java.lang.String
The XML signature as a string.

Throws

None.

setXMLSignature

Description

setXMLSignature sets the XML signature. The XML Signature must be well-formed XML and conform to the official XML Digital Signature Specification.

Syntax

```
public void setXMLSignature(java.lang.String _digitalSignature)
```

Parameters

Name	Type	Description
_digitalSignature	String	The XML signature as a string.

Returns

Void.

Throws

None.

6.13 SOAPSigner Class

The **SOAPSigner** class is a utility node in the SOAP ETD. This class signs key parts of the SOAP message, generating **SOAPSignature** objects.

The **SOAPSigner** class is defined as:

```
public class SOAPSigner
```

The **SOAPSigner** class extends **java.lang.Object.**

The **SOAPSigner** class methods include:

getSignatureResults

Description

getSignatureResults retrieves the results of the last call to sign. Its values can be:

- **Empty string** indicating success.
- **Exception stack trace** indicating failure

Syntax

```
public java.lang.String getSignatureResults()
```

Parameters

None.

Returns

java.lang.String
The signature results.

Throws

None.

getSignatures

Description

getSignatures retrieves the list of signatures.

Syntax

```
public java.util.List getSignatures()
```

Parameters

None.

Returns

List
The list of signatures.

Throws

None.

setSignatureResults

Description

setSignatureResults sets the signature results to **_signatureResults**.

Syntax

```
public void setSignatureResults(java.lang.String _signatureResults)
```

Parameters

Name	Type	Description
_signatureResults	String	The signature results.

Returns

Void.

Throws

None.

setSignatures

Description

setSignatures sets the list of signatures.

Syntax

```
public void setSignatures(java.util.List _signatures)
```

Parameters

Name	Type	Description
_signatures	List	The data structure used to hold signatures; it cannot be null.

Returns

Void.

Throws

None.

sign

Description

sign signs the given **_etd** object. The **_etd** object must be an XML-based node. Moreover, the ETD object must have an identification (ID) attribute or be able to generically set attributes. For example, all SOAP-based nodes allow for generic attributes. After signing a SOAP-based node (for example, **SOAPRequest**, **SOAPResponse**, or **SOAPBody**) a new **ID** attribute is added.

If the **_etd** object is generated from a DTD, then the element represented by **_etd** must have an **ID** attribute. This element has a corresponding **setID** that is used to create the proper references.

Syntax

```
public void sign(com.stc.jcsre.ETD _etd)
```

Parameters

Name	Type	Description
_etd	Object	The ETD object to be signed.

Returns

Void.

Throws

None.

6.14 SOAPTransport Class

The **SOAPTransport** class encompasses all the information and methods needed to send or transport a SOAP request to a SOAP server.

The **SOAPTransport** class is defined as:

```
public class SOAPTransport
```

The **SOAPTransport** class extends **java.lang.Object.**

The **SOAPTransport** class methods include:

getStatusCode

Description

getStatusCode retrieves the result of an HTTP post to the SOAP server.

Syntax

```
public int getStatusCode()
```

Parameters

None.

Returns

Integer
The status code for the last call to **SendToSOAPServer**.

Throws

java.lang.Exception when it is unable to retrieve the code.

getStatusMessage

Description

getStatusMessage retrieves any error messages from the last call to **SendToSOAPServer**.

Syntax

```
public java.lang.String getStatusMessage()
```

Parameters

None.

Returns

java.lang.String
The error message, if it exists.

Throws

java.lang.Exception when it is unable to retrieve the error message.

sendToSOAPServer

Description

sendToSOAPServer sends the SOAP message represented by the **SOAPRequest** node.

Syntax

```
public void sendToSOAPServer()
```

Parameters

The configuration parameters are derived from the SOAP Connection Point.

Name	Type	Description
URL	String	The location to send the message or the identifier into which the SOAP message is posted.
SOAPAction URI	String	The identifier from which the server is told what action to take.
Transport Type	String	The transport mechanism, for example, HTTP, SMTP, or HTTP(S). Currently, HTTP is the only transport mechanism supported.

Returns

Void.

Throws

java.lang.Exception when any generic error occurs.

setStatusCode

Description

setStatusCode sets the status code for this SOAP Transport object.

Syntax

```
public void setStatusCode(int_statusCode)
```

Parameters

Name	Type	Description
_statusCode	Integer	The new status code.

Returns

Void.

Throws

None.

setStatusMessage

Description

getStatusMessage sets the status message for this **SOAPTransport** object.

Syntax

```
public void setStatusMessage(java.lang.String _statusMessage)
```

Parameters

Name	Type	Description
_statusMessage	String	The new status message.

Returns

Void.

Throws

None.

6.15 SOAPVerification Class

The **SOAPVerification** class is a utility class used to verify the SOAP message. This class examines all SOAP signature elements in the SOAP header and verifies that they match, based on the digest and signature algorithms.

The XML signatures must have the **KeyInfo** element. This element gives a representation of the key needed to validate the current document.

By default, verification is not done upon unmarshaling of the SOAP message. You must call the **verify** method within the Collaboration. You must then check the call's status to see whether to accept the message.

The **SOAPVerification** class is defined as:

```
public class SOAPVerification
```

The **SOAPVerification** class extends **java.lang.Object.**

The **SOAPVerification** class methods include:

getVerificationResults

Description

getVerificationResults retrieves the verification results as a string. The verification results can be one of the following values:

- **0**: Both the signature and reference validation were successful.
- **1**: Signature validation failed but the reference validation was successful.
- **2**: Signature validation was successful but the reference validation failed.

If you receive any other value, both the signature and reference validation failed.

Note: *See the official XML Digital Signature Specification for details on signature and reference validation.*

Syntax

```
public java.lang.String getVerificationResults()
```

Parameters

None.

Returns

java.lang.String
The results of the last call to verify.

Throws

None.

setVerificationResults

Description

setVerificationResults sets the verification results to the given string.

Syntax

```
public void setVerificationResults(java.lang.String
    _verificationResults)
```

Parameters

Name	Type	Description
_verificationResults	String	The new results from a call to verify.

Returns

Void.

Throws

None.

verify

Description

verify verifies the signatures in the SOAP header.

Syntax

```
public void verify()
```

Parameters

None.

Returns

Void.

Throws

java.lang.Exception if any error occurs.

Chapter 7

Additional Features

This chapter explains additional features available with the e*Way Intelligent Adapter for SOAP, including:

- Secured Sockets Layer (SSL)
- SOAP message attachments
- Digital signature support

7.1 Using Secured Sockets Layer

SSL is supported through the use of JSSE version 1.0.2. Currently, the JSSE reference implementation is used. JSSE is a provider-based architecture. Essentially, this means that there is a set of standard interfaces for cryptographic algorithms, hashing algorithms, secured-socket-layered URL stream handlers, and so on.

Because the user is interfacing with JSSE through these interfaces, the different components can be mixed and matched as long as the implementation is programmed under the published interfaces. However, some implementation cannot support a particular algorithm.

Note: *See the JSSE documentation provided by Sun Microsystems for further details.*

7.1.1 KeyStores and TrustStores

JSSE makes use of files called KeyStores and TrustStores. A KeyStore is a database consisting of a private key and an associated certificate, or an associated certificate chain. The certificate chain consists of the client certificate and one or more certificate authority (CA) certificates.

A KeyStore contains a private key, in addition to the certificate, while TrustStore only contains the certificates trusted by the client (a "trust" store). The installation of the Java HTTP(S) e*Way installs a TrustStore file named **trustedcacertsjks**. This file can be used as the TrustStore for the e*Way.

A KeyStore is used by the e*Way for client authentication, while a TrustStore is used to authenticate a server in SSL authentication. Both KeyStore and TrustStores are managed via a utility called **keytool**, which is a part of the Java JDK installation.

Note: *To use **keytool**, you must set your CLASSPATH to **jcert.jar, jnet.jar**, and **jsse.jar**.*

The following line must also be added to the **jre\lib\security\java.security**:

```
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
```

See the installation manual for the JSSE version 1.0.2 for more information.

7.1.2 Methods for generating a KeyStore and TrustStore

This section explains steps on how to create a KeyStore and a TrustStore (or import a certificate into an existing TrustStore such as **trustedcacertsjks**). The primary tool used is **keytool**, but **openssl** is also used as a reference for generating **pkcs12** KeyStores. For more information on **openssl**, and available downloads, see the following Web site:

http://www.openssl.org.

Creating a TrustStore

For demonstration purposes, suppose you have the following CAs that you trust: **firstCA.cert, secondCA.cert, thirdCA.cert**, located in the directory **C:\cascerts**. You can create a new TrustStore consisting of these three trusted certificates.

To create a new TrustStore

Use the following command:

```
keytool -import -file C:\cascerts\firstCA.cert -alias firstCA
    -keystore myTrustStore
```

You must enter this command two more times, but for the second and third entries, substitute **secondCA** and **thirdCA** for **firstCA**. Each of these command entries has the following purposes:

1. The first entry creates a KeyStore file name **myTrustStore** in the current working directory and imports the **firstCA** certificate into the TrustStore with an alias of **firstCA**. The format of **myTrustStore** is JKS.
2. For the second entry, substitute **secondCA** to import the **secondCA** certificate into the TrustStore, **myTrustStore**.
3. For the third entry, substitute **thirdCA** to import the **thirdCA** certificate into the TrustStore.

Once completed, myTrustStore is available to be used as the TrustStore for the e*Way. See **"TrustStore" on page 37** for more information.

Using an Existing TrustStore

This section explains how to use an existing TrustStore such as trustedcacertsjks. Notice that in the previous section, steps 2 and 3 were used to import two CAs into the TrustStore created in step 1.

For example, suppose you have a trusted certificate file named: **C:\trustedcerts\foo.cert** and want to import it to the **trustedcacertsjks** TrustStore.

If you are importing certificates into an existing TrustStore, use:

```
keytool -import -file C:\cacerts\secondCA.cert -alias secondCA
    -keystore trustedcacertsjks
```

Once you are finished, **trustedcacertsjks** can be used as the TrustStore for the e*Way. See **"TrustStore" on page 37** for more information.

7.1.3 Creating a KeyStore in JKS Format

This section explains how to create a KeyStore using the JKS format as the database format for both the private key, and the associated certificate or certificate chain. By default, as specified in the java.security file, **keytool** uses JKS as the format of the key and certificate databases (KeyStore and TrustStores). A CA must sign the certificate signing request (CSR). The CA is therefore trusted by the server-side application to which the e*Way is connected.

To generate a KeyStore

Use the following command:

```
keytool -keystore clientkeystore -genkey -alias client
```

You are prompted for several pieces of information required to generate a CSR. A sample key generation section follows:

```
Enter keystore password: seebyond
What is your first and last name?
[Unknown]: development.seebeyond.com
What is the name of your organizational unit?
[Unknown]: Development
what is the name of your organization?
[Unknown]: SeeBeyond
What is the name of your City of Locality?
[Unknown]: Monrovia
What is the name of your State or Province?
[Unknown]: California
What is the two-letter country code for this unit?
[Unknown]: US
Is<CN=Foo Bar, OU=Development, O=SeeBeyond, L=Monrovia,
ST=California, C=US> correct?
[no]: yes

Enter key password for <client>
    (RETURN if same as keystore password):
```

If the KeyStore password is specified, then the password must be provided for the e*Way. Press RETURN when prompted for the key password (this action makes the key password the same as the KeyStore password).

This operation creates a KeyStore file **clientkeystore** in the current working directory. You must specify a fully-qualified domain for the "first and last name" question. The sample uses **development.seebeyond.com**. The reason for this use is that some CAs such as Verisign expect this parameter to be a fully qualified domain name.

There are CAs that do not require the fully qualified domain, but it is recommended to use the fully-qualified domain name for the sake of portability. All the other information given must be valid. If the information can not be validated, Certificate Authority such as Verisign does not sign a generated CSR for this entry.

This KeyStore contains an entry with an alias of **client**. This entry consists of the Generated private key and information needed for generating a CSR as follows:

```
keytool -keystore clientkeystore -certreq alias client -keyalg rsa
    -file client.csr
```

This command generates a certificate signing request which can be provided to a CA for a certificate request. The file **client.csr** contains the CSR in PEM format.

Some CA (one trusted by the Web server to which the e*Way is connecting) must sign the CSR. The CA generates a certificate for the corresponding CSR and signs the certificate with its private key. For more information, visit:

http://www.thawte.com

or

http://www.verisign.com

If the certificate is chained with the CA's certificate, perform step A; otherwise, perform step B in the following list:

A The following command assumes the client certificate is in the file **client.cer** and the CA's certificate is in the file **CARoot.cer**:

```
keytool -import -keystore clientstore -file client.cer -alias
    client
```

This command imports the certificate (which can include more than one CA in addition to the Client's certificate).

B The following command imports the CA's certificate into the KeyStore for chaining with the client's certificate:

```
keytool -import -keystore clientkeystore -file CARootcer -alias
    theCARoot
```

C The following command imports the client's certificate signed by the CA whose certificate was imported in the preceding step:

```
keytool -import -keystore clientkeystore -file client.cer -alias
    client
```

The generated file **clientkeystore** contains the client's private key and the associated certificate chain used for client authentication and signing. The KeyStore and/or **clientkeystore**, can then be used as the e*Way's KeyStore. See the **"KeyStore" on page 36** for more information.

7.1.4 Creating a KeyStore in PKCS12 Format

This section explains how to create a PKCS12 KeyStore to work with JSSE. In a real working environment, a customer could already have an existing private key and certificate (signed by a known CA). In this case, JKS format can not be used, because it does not allow the user to import/export the private key through **keytool**. It is necessary to generate a PKCS12 database consisting of the private key and its certificate.

The generated PKCS12 database can then be used as the e*Way's KeyStore. The **keytool** utility is currently lacking the ability to write to a PKCS12 database. However, it can read from a PKCS12 database.

Note: *There are additional third-party tools available for generating PKCS12 certificates, if you want to use a different tool.*

For the following example, **openssl** is used to generate the PKCS12 KeyStore:

```
cat.mykey.pem.txt mycertificate.pem.txt>mykeycertificate.pem.txt
```

The existing key is in the file **mykey.pem.txt** in PEM format. The certificate is in **mycertificate.pem.txt**, which is also in PEM format. A text file must be created which contains the key followed by the certificate as follows:

```
openssl pkcs12 -export -in mykeycertificate.pem.txt -out
    mykeystore.pkcs12 -name myAlias -noiter -nomaciter
```

This command prompts the user for a password. The password is required. The KeyStore fails to work with JSSE without a password. This password must also be supplied as the password for the e*Way's KeyStore password (see **"KeyStorePassword" on page 37**).

This command also uses the **openssl pkcs12** command to generate a PKCS12 KeyStore with the private key and certificate. The generated KeyStore is **mykeystore.pkcs12** with an entry specified by the **myAlias** alias. This entry contains the private key and the certificate provided by the **-in** argument. The **noiter** and **nomaciter** options must be specified to allow the generated KeyStore to be recognized properly by JSSE.

7.1.5 SSL Handshaking

There are two options available for setting up SSL connectivity with a Web server:

- **Server-side authentication:** The majority of eCommerce Web sites on the Internet are configured for server-side authentication. The e*Way requests a certificate from the Web server and authenticates the Web server by verifying that the certificate can be trusted. Essentially, the e*Way does this operation by looking into its TrustStore for a CA certificate with a public key that can validate the signature on the certificate received from the Web server.
- **Dual authentication:** This option requires authentication from both the e*Way and Web server. The server side (Web server) of the authentication process is the same as that described previously. However, in addition, the Web server requests a certificate from the e*Way. The e*Way then sends its certificate to the Web server. The server, in turn, authenticates the e*Way by looking into its TrustStore for a matching trusted CA certificate. The communication channel is established by the process of both parties' requesting certificate information.

For illustrations of both these types of authentication, see the following figures:

- **Figure 24 on page 134** shows a diagram of the SSL handshake dialog for server-side authentication.
- **Figure 25 on page 135** shows a diagram of the SSL handshake dialog for dual authentication.

Figure 24 Server-side Authentication

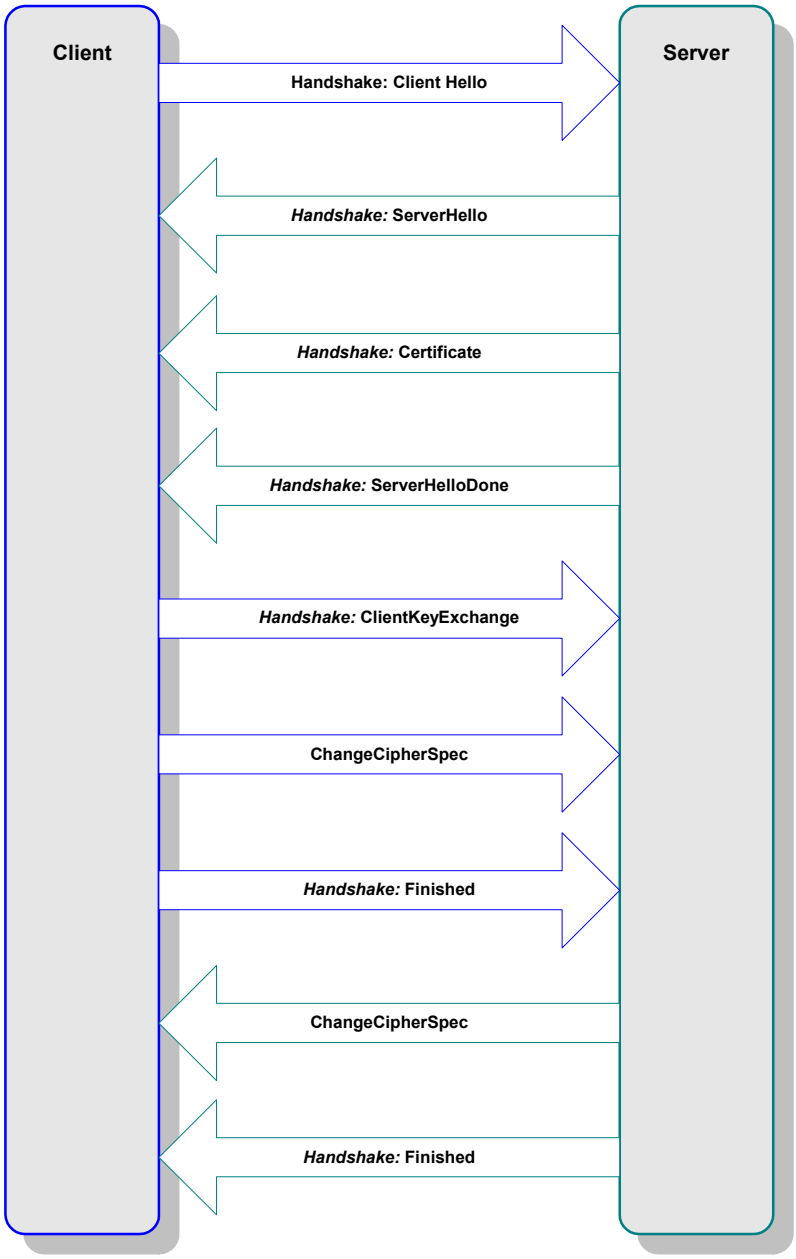


Figure 25 Dual Authentication

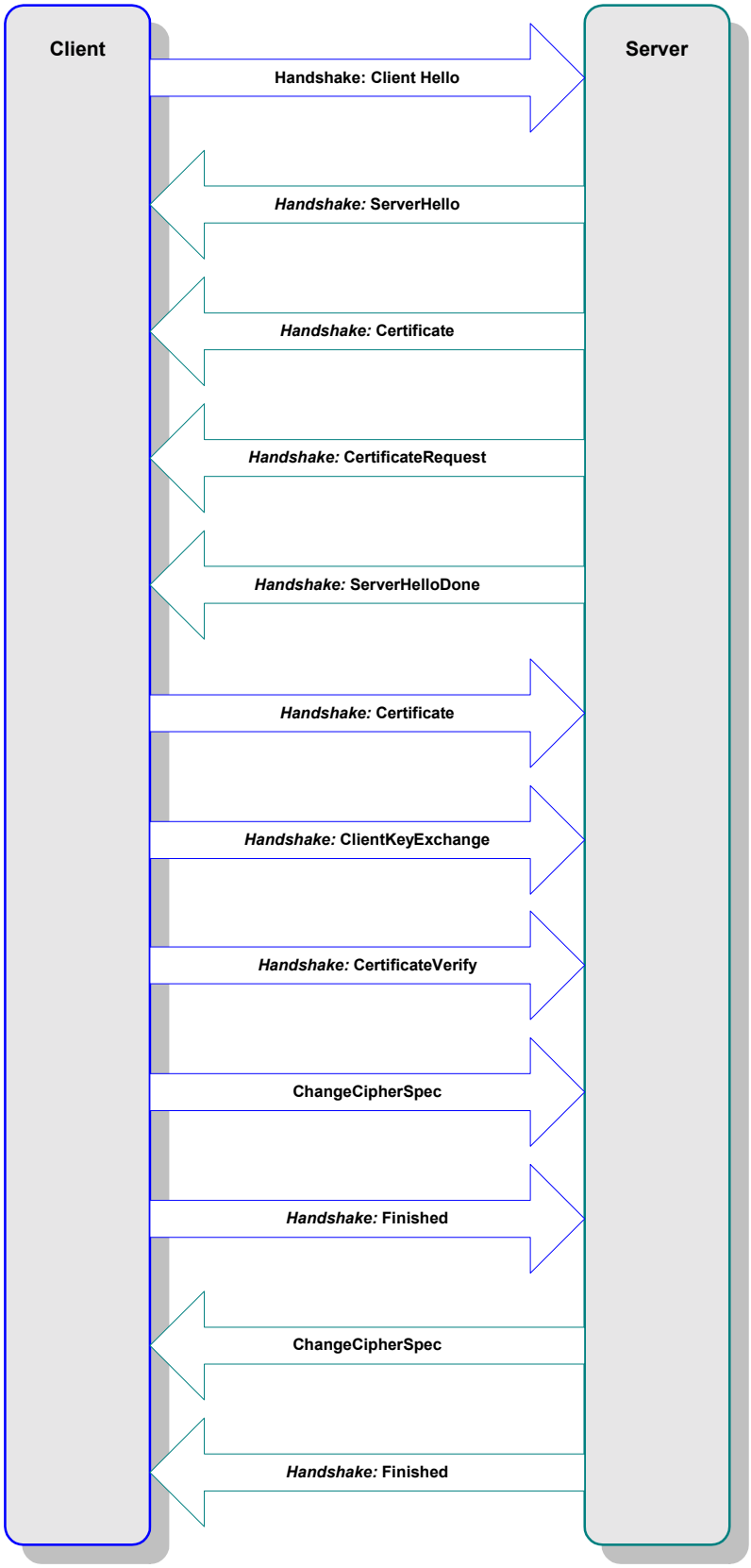
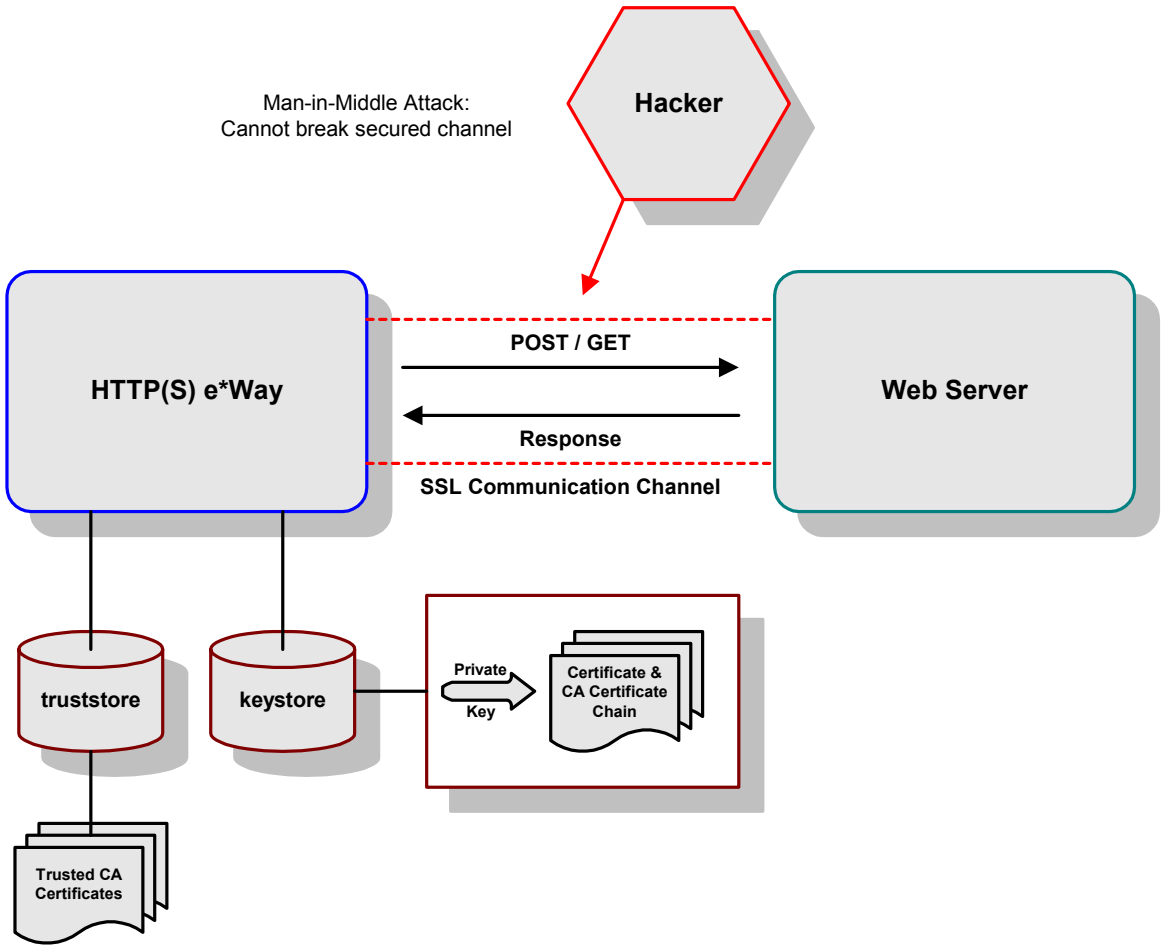


Figure 26 shows a diagram of general SSL operation with the HTTP(S) e*Way.

Figure 26 General SSL Operation: HTTP(S) e*Way



Note: *See the **HTTP(S) e*Way Intelligent Adapter User's Guide** for details on how to use this e*Way.*

7.2 Using SOAP Attachments

You can send a SOAP message together with attachments of various sorts, ranging from fax images to art drawings or images. These attachments are usually transmitted in some type of binary format. For example, most images on the Internet are transmitted using either the .**gif** or .**jpg** file data formats.

7.2.1 SOAP Attachments: Overview

Web service developers can specify that a Web service's methods are to use SOAP attachments to transport binary data or large Extensible Markup Language (XML) documents.

One typical use of SOAP attachments is for transporting intact, binary data such as image files. For another example, your system may need to transport XML documents to other parts of the system without the overhead of validating them. These XML documents that do not need to conform to your particular schema or Document Type Definition (DTD) can be passed as attachments.

By using SOAP attachments, the SOAP message body is much smaller because it contains only a reference to the data and not the data itself. Using attachments can be more efficient because smaller SOAP messages are processed more quickly than extremely large messages, and the translation of the data to Java objects is reduced for attachments.

SOAP Attachment Implementation

SOAP attachments are implemented by wrapping the SOAP message and one or more attachments in an envelope of Multipurpose Internet Mail Extensions (MIME). The developer uses, for example, the Web Service Builder (or an equivalent) to map method parameters or return value data types to MIME types.

Table 6 shows the Java data types that can be set to use SOAP attachments. The table also shows the MIME type that each Java type can be mapped to.

Table 6 Appropriate Mappings for Java Types to MIME Types

Java Data Type	MIME Type Mapping
java.lang.String	text/plain, text/xml
org.w3c.dom.Document	text/xml
byte[]	image/gif, image/jpeg, application/octet-stream
java.io.Serializable	application/x-java-serialized-object

To make a Web service use SOAP attachments

1. From the Web Service Builder (for example), select the method parameter or return value for which you want to use attachments.
2. Place a check next to **Part is Attachment**, if it is available.
3. Select from the list of **Available Types** an appropriate MIME type you want the application to use as an attachment.
4. Click **Add Type**.
5. If you want more than one type of attachment for your application, Repeat steps 2 and 3 for more types, as necessary. For example, you can allow attachments of both **image/gif** and **image/jpeg** MIME types.

After the Web service is deployed and the method that uses attachments is invoked, the MIME attachment is translated to a Java object by an appropriate data content handler, depending on the MIME data type. For example, the **text/xml MIME** type is translated to a **w3c.dom.Document** object.

7.2.2 Associating SOAP Messages and Attachments

This section explains a standard way to associate a SOAP message with one or more attachments in their native format in a multipart MIME structure for transport. The specification combines a specific usage of **Multipart/Related** MIME media type and the URI schemes explained on the World Wide Web Consortium (W3C) Web site, for referencing MIME parts.

For additional details, see the SOAP-related pages of the W3C Web site at the following URL:

http://www.w3c.org

The processes explained in this section treat the multipart MIME structure as essentially a part of the transfer protocol binding, that is, on par with the transfer protocol headers, as far as the SOAP message is concerned. This multipart structure is called the SOAP message package.

The purpose of this section is to show how to use existing facilities in SOAP, as well as standard MIME mechanisms, to carry and reference attachments, using existing standards and without inventing ways on your own. Most Internet communication protocols can transport MIME-encoded content, although some special considerations are required for the Hyper-text Transfer Protocol (HTTP).

Note: *For more information, see* **"HTTP Binding" on page 144**.

SOAP Message Packages

A SOAP message package contains a primary SOAP 1.1 message. It can also contain additional elements not contained within the SOAP message but related in some way. These elements can contain data in formats other than XML.

The primary SOAP 1.1 message in a message package can reference these additional elements, called attachments. This section explains how to construct SOAP message packages and how SOAP processors handle them.

SOAP Message Package Construction

A SOAP message package is constructed using the **Multipart/Related** media type. The basic rules for constructing SOAP message packages are:

- The primary SOAP 1.1 message must be carried in the root body part of the **Multipart/Related** structure. Consequently, the **type** parameter of the **Multipart/Related** media header is always the same as the **Content-Type** header for the primary SOAP 1.1 message, that is, **text/xml**.
- Referenced MIME parts must contain either a **Content-ID** MIME header or a **Content-Location** MIME header.

Be sure that the root part contains a **Content-ID** MIME header. In addition to the required parameters for the **Multipart/Related** media type, the start parameter must always be present. This construction permits more robust error detection.

For example, a SOAP processor compliant with this specification receives a SOAP 1.1 message (carried in the root body part of a **Multipart/Related** MIME message). The processor must therefore handle the SOAP message according to the rules for processing SOAP 1.1 messages as defined by SOAP 1.1. Also, a SOAP processor that receives an invalid message must generate a client fault code compliant with SOAP 1.1.

Attaching to SOAP Messages

The MIME **Multipart/Related** encapsulation of a SOAP message is semantically equivalent to a SOAP protocol binding. The SOAP message is not aware that it is being encapsulated. There is nothing in the primary SOAP message itself to indicate the message is encapsulated.

SOAP 1.1 Message and Attachment Example

This example shows a SOAP 1.1 message with an attached fax image of a signed form in .**tiff** file format:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/
xml;
        start="<claim061400a.xml@claiming-it.com>"
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
..
<theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
..
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.tiff@claiming-it.com>

...binary TIFF image...
--MIME_boundary--
```

Note: *In these examples the **Content-Type** header line has been continued across two lines so the example fits easily on a page. SOAP message senders must transmit headers on a single line.*

SOAP References to Attachments

Both the header entries and body of the primary SOAP 1.1 message can be required to refer to other elements in the message package. This section explains how to accomplish this process using existing mechanisms in SOAP and MIME.

The data-encoding rules given in SOAP 1.1 allow the value of an accessor to be given by reference. In other words, it can be given as a resource referenced by a URI provided as the value of an **href** attribute. The SOAP encoding schema allows the value of an **href** attribute to be any URI reference. The attribute can therefore be used to reference not just XML fragments within a SOAP 1.1 message, but any resource.

Resolution Process

This specification describes a usage pattern of the SOAP **href** attribute in SOAP 1.1 to allow attribute values to be references to attachments carried as MIME parts in the SOAP message package. The resolution process for URI references (including references used in **href** attributes) in the primary SOAP 1.1 message happens under the following conditions:

- It occurs within a SOAP message package.
- It is based on the rules for multipart MIME messages with **text/html** root documents.

These rules are adapted from the HTML and rendering context and applied to the SOAP 1.1 messaging context.

The resolution process operates as follows:

- Conversion of all URI references to absolute references
- Resolution of the absolute references

The W3C SOAP 1.1 specifications provide rules for both steps (see the W3C Web site for additional information; also see **"Conventions and Specifications" on page 9**).

Note: *This process does not apply to same-document references.*

The semantics of the SOAP 1.1 pattern that involves using an **href** attribute with a fragment identifier remains unchanged if the following conditions are present:

- The fragment identifier references an XML element in the same SOAP 1.1 message.
- That SOAP message is based on a label defined by an **ID** attribute.

Converting Relative URI References to Absolute

The authoritative process for converting relative URI references to absolute references is defined in the W3C SOAP 1.1 specifications (see the W3C Web site). Before this process can happen, the base URI must be established.

The specifications for establishing a base URI follow these essential rules:

1. **Base URI within Document Content:** The mechanism for explicit specification of a base URI within a SOAP 1.1 message is the XML base mechanism.

2 **Base URI from an Encapsulating Entity:**

- If there is a **Content-Location** header containing an absolute URI in any MIME element enclosing the primary SOAP 1.1 message,
- The URI from the closest such **Content-Location** header is the base URI for the element.

3 **Base URI from the Retrieval URI:** The retrieval URI for a SOAP message package is never allowed to be used as a base URI.

4 **Default Base URI:** The default base URI is **thismessage:/** in accordance with W3C SOAP 1.1 specifications.

Note: *The previous rules are shown in order of precedence.*

Every MIME part in the **Multipart/Related** structure that constitutes a SOAP message package has at least one absolute URI label. The following list shows the types of URI labeling:

- If a **Content-Location** header is present with an absolute URI value, that URI is a label for the part.
- If a **Content-Location** header is present with a relative URI value, you must apply rules 2 and 4 in the previous list, to establish the base URI. Use this URI for the process of converting the relative URI to an absolute one. The resulting absolute URI is a label for the part.
- If a **Content-ID** header is present, an absolute URI label for the part is formed using the CID URI scheme (see the W3C Web site).

The resolution of absolute URI references operates as follows:

1 For each referencing URI in the primary SOAP 1.1 message, compare:

- The value of the referencing URI, after conversion to the absolute form as described previously
- With the URI labels derived from **Content-ID** and **Content-Location** headers for other body parts in the surrounding **Multipart/Related** structure

Note: *The rules for URI comparison are given in the W3C SOAP 1.1 specifications (see the W3C Web site).*

2 If a match is found, the element contained in the MIME part is the referant. If no match is found, use normal resolution rules based on the URI scheme.

3 In case of conflicting labels based on **Content-ID** and **Content-Location** headers, use the rules given in the W3C SOAP 1.1 specifications to resolve the conflict.

Example of SOAP Message with Absolute URI Referencing

The example shown under **"SOAP 1.1 Message and Attachment Example" on page 139** illustrates the use of the CID reference in the body of the SOAP 1.1 message. Here is the same example now rewritten using absolute URIs referencing elements labeled using **Content-Location** headers:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/
xml;
        start="<http://claiming-it.com/claim061400a.xml>"
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <http://claiming-it.com/claim061400a.xml>
Content-Location: http://claiming-it.com/claim061400a.xml

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
..
<theSignedForm href="http://claiming-it.com/claim061400a.tiff"/>
..
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <http://claiming-it.com/claim061400a.tiff>
Content-Location: http://claiming-it.com/claim061400a.tiff

...binary TIFF image...
--MIME_boundary--
```

Examples of SOAP Message with Relative URI Referencing

Here is the same example, this time using relative URIs that use the **Content-Location** header at the base of the MIME **Multipart/Related** structure for their base URI:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/
xml;
        start="<http://claiming-it.com/claim061400a.xml>"
Content-Description: This is the optional message description.
Content-Location: http://claiming-it.com/

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <http://claiming-it.com/claim061400a.xml>
Content-Location: claim061400a.xml

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
..
<theSignedForm href="claim061400a.tiff"/>
..
```

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-Location: claim061400a.tiff

...binary TIFF image...
--MIME_boundary--
```

In addition, here is an example that uses relative URIs but no explicit base URI, so rule 4 under **"Converting Relative URI References to Absolute" on page 140**, for establishing a base URI applies. Using this rule causes relative URIs in the SOAP message and **Content-Location** labels to use the base URI of **thismessage:/** as follows:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/
xml;
        start="<b6f4ccrt@15.4.9.92/s445>"
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <b6f4ccrt@15.4.9.92/s445>
Content-Location: claim061400a.xml

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
..
<theSignedForm href="the_signed_form.tiff"/>
..
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <a34ccrt@15.4.9.92/s445>
Content-Location: the_signed_form.tiff

...binary TIFF image...
--MIME_boundary-
```

Determining URI Resolution

Within a SOAP message, the fact that a URI reference occurs as the value of a SOAP **href** attribute does not by itself imply that the receiving SOAP processor must resolve the URI. It is up to the SOAP processor to determine whether resolution of the URI is required. This determination is based on the processing semantics of the message.

The receiving SOAP processor can choose to ignore the URI even if it is referencing a MIME attachment. Conversely, all attachments that appear in the SOAP message package cannot be referenced in the root SOAP message.

Note: *This section does not provide a means, within a SOAP message, to explicitly mark it as the root of a SOAP message package. For example, this marking could be with a distinguished header entry that enumerates message package contents. This specification defines an extension to the transport binding mechanisms defined in the SOAP 1.1 specifications. See the W3C Web site for details.*

Relationship to SOAP 1.1

The packaging of a SOAP 1.1 message in the root part of a **Multipart/Related** MIME structure (along with other content) can be viewed as a specific method for carrying SOAP 1.1 messages in any protocol capable of transferring MIME-encoded content.

A SOAP processor capable of supporting both the MIME-based encoding described here and the base transport over which it is carried, must:

- Treat the SOAP 1.1 message in the root part as the message to be processed
- Follow all the rules of SOAP 1.1 for the SOAP 1.1 message and for the base transport binding used (see the example of HTTP binding described in section 6 of the SOAP 1.1 specifications).

See **"HTTP Binding"** (next section) for more information on that subject.

HTTP Binding

This section explains the rules for carrying a compound SOAP message in an HTTP message. As in the case of the base SOAP 1.1 specification, this specification does not prescribe either an asynchronous messaging or a synchronous request/response interaction pattern.

Instead, this description of the HTTP binding explains the relationship between HTTP headers and the MIME headers used in constructing a SOAP message package, without restricting the interaction pattern in any way.

Rules for Using SOAP with HTTP

The basic approach to carrying multipart MIME structure in an HTTP message, in this specification, is:

- To confine MIME-encoded content to the MIME parts and
- Use the multipart media type header at the HTTP level as a native HTTP header

The rules for forming an HTTP message containing a SOAP message package are:

- The **Content-Type**: **Multipart/Related** MIME header must appear as an HTTP header. The rules for parameters of this header specified under **"SOAP Message Packages" on page 138** also apply here.
- No other headers with semantics defined by MIME specifications (such as **Content-Transfer-Encoding**) are permitted to appear as HTTP headers. Specifically, the "MIME-Version: 1.0" header must not appear as an HTTP header. Note that HTTP itself uses many MIME-like headers with semantics defined by HTTP 1.1. These headers are allowed.
- The MIME parts containing the SOAP message and the attachments constitute the HTTP message's body and must appear as described under **"SOAP Message Packages" on page 138**, including appropriate MIME headers.

Unlike within HTTP, MIME semantics apply at the SMTP message level. Therefore for SMTP transport, the multipart MIME headers can simply merge with the SMTP headers.

Example of HTTP Message with SOAP Message/Attachments

This example shows an HTTP message containing a SOAP message package including two attachments. The SOAP 1.1 message contains relevant information and is sent with a fax image of a signed form (.**tiff** file) and a digital photo (.**jpeg** file). The example follows:

```
POST /insuranceClaims HTTP/1.1
Host: www.risky-stuff.com
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/
xml;
        start="<claim061400a.xml@claiming-it.com>"
Content-Length: XXXX
SOAPAction: http://schemas.risky-stuff.com/Auto-Claim
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<claim:insurance_claim_auto id="insurance_claim_document_id"
xmlns:claim="http://schemas.risky-stuff.com/Auto-Claim">
<theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
<theCrashPhoto href="cid:claim061400a.jpeg@claiming-it.com"/>
<!-- ... more claim details go here... -->
</claim:insurance_claim_auto>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim061400a.tiff@claiming-it.com>
```

```
...Base64 encoded TIFF image...
--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.jpeg@claiming-it.com>

...Raw JPEG image..
--MIME_boundary--
```

Note: *As in the previous examples, the* ***Content-Type*** *header line has been continued across two lines to fit easily on a page. Again, SOAP message senders must transmit headers on a single line.*

7.3 Using Digital Signatures

The purpose of digital signatures is to allow the recipient of a data object, usually a message, to verify the data's authenticity. Digital signatures are a value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity.

This section explains the syntax and processing rules of a SOAP header entry to carry digital signature information within a SOAP 1.1 envelope.

7.3.1 Header Entry Syntax

This section explains the syntax for SOAP header entry.

Namespace

The XML namespace **[XML-ns]** URI that must be used by implementations of this specification and can be found at the following URI:

http://schemas.xmlsoap.org/soap/security/2000-12

The namespace prefix **SOAP-SEC** used in this specification is associated with this URI.

Signature Header Entry

The header entry **<SOAP-SEC:Signature>** is defined by the following schema:

[XML-Schema1], [XML-Schema2]

The **<SOAP-SEC:Signature>** element contains a single digital signature conforming to the XML-signature specification **[XML-Signature]** as follows:

```
<schema
  xmlns="http://www.w3.org/1999/XMLSchema"
  xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
  targetNamespace="http://schemas.xmlsoap.org/soap/security/2000-12"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <import namespace="http://www.w3.org/2000/09/xmldsig#"/>
  <import namespace="http://schemas.xmlsoap.org/soap/envelope/"/>

  <element name="Signature" final="restriction">
    <complexType>
      <sequence>
        <element ref="ds:Signature" minOccurs="1" maxOccurs="1"/>
      </sequence>
      <attribute name="id" type="ID" use="optional"/>
      <attribute ref="SOAP-ENV:actor" use="optional"/>
      <attribute ref="SOAP-ENV:mustUnderstand" use="optional"/>
    </complexType>
  </element>

  <attribute name="id" type="ID"/>

</schema>
```

SOAP-SEC:id Attribute

The **<ds:Reference>** element *must* refer to the signed part of the SOAP envelope. This reference can be achieved through the use of XML identifiers. Applications are responsible for determining which attributes are of the type **ID**.

To help applications identify attributes of the type **ID**, this specification defines the **SOAP-SEC:id** global attribute. This attribute can be used for referencing the signed part of the SOAP envelope.

Example of SOAP Message with Signature Header

Here is an example of a SOAP message with a signature header entry, where the SOAP body is signed and the resulting signature **<ds:Signature>** is added to the **<SOAP-SEC:Signature>** header entry. Note that the **URI** attribute of the **<ds:Reference>** element refers to the **<SOAP-ENV:SOAP-Body>** element. The example follows:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <SOAP-SEC:Signature
      xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-
12"
      SOAP-ENV:actor="some-URI"
      SOAP-ENV:mustUnderstand="1">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-
20001026">
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#dsa-sha1"/>
          <ds:Reference URI="#Body">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/TR/2000/CR-
xml-c14n-20001026"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/
xmldsig#sha1"/>
```

```
            <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</
ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>MC0CFFrVLtRlk=...</ds:SignatureValue>
      </ds:Signature>
    </SOAP-SEC:Signature>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body
    xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
    SOAP-SEC:id="Body">
    <m:GetLastTradePrice xmlns:m="some-URI">
      <m:symbol>IBM</m:symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7.3.2 Processing Rules

The **Signature** header entry is used to carry a signature compliant with the XML-signature specification **[XML-Signature]** within a SOAP envelope. It can be used for signing one or more elements in the SOAP envelope. Multiple signature header entries may be added into a single SOAP envelope with either disjoint or overlapping signed elements.

Note: *A future version of this specification may allow signature syntax other than using **[XML-Signature]**, through extension **[XML-Schema1]** of the content model of **<SOAP-SEC:Signature>**.*

SOAP applications conforming to this specification *must* satisfy the following conditions:

- The application must be capable of processing an XML signature as defined in the XML-signature specification **[XML-Signature]**.
- If a conforming SOAP application is to add a **<SOAP-SEC:Signature>** header entry in the SOAP Header, the header entry must have a **<ds:Signature>** element conforming to **[XML-Signature]**. All the **<ds:Reference>** elements contained in the signature must refer to a resource within the enclosing SOAP envelope or to a resource in the enclosing SOAP message package **[SOAP-attachment]** if the envelope is the primary SOAP 1.1 message **[SOAP-attachment]** of the package.
- When a conforming SOAP application receives a SOAP message containing one or more **<SOAP-SEC:Signature>** header entries intended for the application (either it is explicitly specified by the SOAP **actor** attribute, or the application is the ultimate destination), for each such header entry, the application must perform the following steps:
  - Decide whether to process the header entry (either forced by the **mustUnderstand="1"** attribute or voluntarily).
  - If it is to be processed, the application must try to validate the signature using the processing model of **[XML-Signature]**.

The XML canonicalization **[XML-C14N]** of **<ds:SignedInfo>** and other signed resources must each be done within its own context. In other words, the canonical form **[XML-C14N]** of **<ds:SigndInfo>** always inherits the namespace declarations for **SOAP-ENV** and **SOAP-SEC**.

Signature Header Entry Generation

This section explains the operations you must perform for the signature header entry.

To create a <SOAP-SEC:Signature> header

1. Prepare the target SOAP envelope with the body and necessary headers.
2. Create a template of a **<ds:Signature>** element. The template is assumed to contain empty contents for **<ds:DigestValue>** or **<ds:SignatureValue>** elements, but contains appropriate values for the elements such as **<ds:SignatureMethod>** and **<ds:Reference>** required to calculate them.
3. Create a new header entry **<SOAP-SEC:Signature>** and add the template to this entry.
4. Add the header entry **<SOAP-SEC:Signature>** to the SOAP header.
5. Add the SOAP **actor** and **mustUnderstand** attributes to the entry, if necessary.
6. Calculate the **<ds:DigestValue>** and **<ds:SignatureValue>** elements according to the core generation of the XML-signature specification **[XML-Signature]**.

XPath filtering can be used to specify objects to be signed, as described in **[XML-Signature]**. However, since the SOAP message exchange model allows intermediate applications to modify the envelope (add or delete a header entry, for example), **XPath** filtering does not always result in the same objects after message delivery.

Take care in using **XPath** filtering so there is no subsequent validation failure because of such modifications. To do so, use the following transform:

http://www.w3.org/2000/09/xmldsig#enveloped-signature

It is defined in **[XML-Signature]** and is useful when signing the entire envelope, including other header entries (if any).

Signature Header Entry Validation

The validation of a **<SOAP-SEC:Signature>** header entry fails under the following conditions:

- The syntax of the content of the header entry does not conform to this specification.
- The validation of the signature contained in the header entry fails according to the core validation of the XML-signature specification **[XML-Signature].**
- The receiving application program rejects the signature for one of its own reasons (for example, the signature is created by an untrusted key).

Note: *If the validation of the signature header entry fails, applications usually report the failure to the sender. See the W3C SOAP 1.1 specifications for information on how to deal with validation failures.*

Security Considerations

The specifications provided in this section define the use of **[XML-Signature]** in SOAP 1.1 headers. As one of the building blocks for securing SOAP messages, **[XML-Signature]** is intended to be used in conjunction with other security techniques. Digital signatures must be understood in the context of other security mechanisms and possible security threats, then used accordingly.

For example, digital signatures alone do not provide message authentication. You can record a signed message and resend it (replay attack). To prevent this type of attack, digital signatures must be combined with an appropriate means to ensure the uniqueness of the message, for example, nonces or time stamps. One way to add this information is to place an extra **<ds:Object>** element as a child of the **<ds:Signature>**.

When digital signatures are used for verifying the identity of the sending party, the sender must prove the possession of the private key. One way to achieve this proof is to use a challenge-response type of protocol.

Implementers must also be aware of all the security implications resulting from the use of digital signatures in general and **[XML Signature]** in particular. In building trust into an application based on a digital signature, the following additional pieces of technology must be identified in relation to the signature:

- Well-defined certificate trust model, whether hierarchical or peer-to-peer
- Generation and maintenance of trusted key pairs and certificates
- Validation that a certificate has not been revoked

Index

J

L

M

O

P

R

S

U

V

W