SeeBeyond ICAN Suite

HTTP(S) e*Way Intelligent Adapter User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version

Version 20050406092250.

Contents

Chapter 7

Java Classes and Methods 76

Chapter 1

Introduction

This guide explains the SeeBeyond™ Technology Corporation's (SeeBeyond™) HTTP(S) e*Way™ Intelligent Adapter. This chapter provides an introduction to the guide and the e*Way.

1.1 HTTP(S) e*Way: Overview

The HTTP(S) e*Way allows e*Gate Integrator to communicate with client applications over the Internet using the HyperText Transfer Protocol (HTTP) and HTTP over Secure Sockets Layer (SSL), that is, HTTP(S).

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).*

The HTTP(S) e*Way supports both the **GET** and **POST** methods. This version of the HTTP(S) e*Way is enabled by the Java programming language.

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 HTTP and HTTP(S) protocol and to be thoroughly familiar with Windows-style operations

1.3 Using the e*Way

This section explains essential information you need to know to use the HTTP(S) e*Way.

1.3.1 Basic Information

The following information applies to the e*Way:

- Supports HTTP versions 1.0 and 1.1
- Adheres to RFCs 1945 (version 1.0), 2616 (version 1.1), and 2817 (TLS over version 1.1)
- Acts as client only
- Supports single-session request/reply scenarios in the default configuration

1.3.2 GET and POST Methods

The **GET** method can be used to retrieve a page specified by the URL or to retrieve information from a form-based Web page by submitting URL-encoded key and name value pairs. In the latter case, the page must support the **GET** method.

The following example shows a URL-encoded query string:

http://google.yahoo.com/bin/query?p=seebeyond+integrator

The URL specifies the search page and the name-value pair for the search. The question mark (?) indicates the beginning of the name-value pair encoding. In the previous example, the name portion of the query is "p," and the value to search is "seebeyond integrator." A query can consist of one or more of these name-value pairs.

Note: *See the HTTP Specification for complete information.*

The **POST** method is more versatile, in that it supports form-based requests, as well as sending large amounts of data. The **POST** method does not have the size-limitation maximum of 255 or 1024 characters (depending on the Web server), which the **GET** method has. As with **GET**, the Web page must support the **POST** method in order to use **POST**.

Taking the previous URL as an example, if you specify the following URL:

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

Then, you can specify the name-value pair separately. The HTTP client allows for the specification of the URL and n-number of value pairs via its methods.

The e*Way also supports automatic URL redirection. Automatic redirection occurs when the e*Way receives a 300 status code, specifically 301.

The HTTP(S) portion of the e*Way is currently handled through the use of Java Server Socket Extension (JSSE) 1.0.2. The reference implementation of JSSE is used and described in this guide.

An HTTP message has basically two parts: a request and a response. The message header is composed of a header line, header fields, a blank line, and an optional body (or data payload). The response is made up of a header line, header fields, a blank line, and an optional body (or data payload). HTTP is a synchronous protocol, that is, a client makes a request to a server and the server returns the response on the same socket.

1.3.3 Sample HTTP Exchange

To retrieve the file at the following URL:

http://www.myhost.com/path/file.html

First open a socket to the host **www.myhost.com**, port 80 (use the default port of 80 because none is specified in the URL). You can then send a request through a socket that looks like the following example:

```
GET /path/file.html HTTP/1.0  (Request Header Line)
User-Agent: HTTP(S)eWay/5.0.1 (Request Header field)
```

The server sends a response back through the same socket. The response could look like the following example:

```
HTTP/1.0 200 OK                (Response Header Line)
Date: Fri, 31 Dec 1999 23:59:59 GMT(Response Header Field)
Content-Type: text/html   (Response Header Field)
Content-Length: 1354           (Response Header Field)
[blank line here]
<html>                         (Response payload)
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
  .
  .
  .
</body>
</html>
```

After sending the response, the server closes the socket.

1.3.4 Support of Cookies

Cookies are also supported by the HTTP(S) e*Way. Essentially a cookie is an HTTP header, which is a key-value pair in the header fields section of an HTTP message.

With regards to cookies, the headers used are **Set-Cookie** and **Cookie**. The **Cookie-request** header is sent from the server in request for cookies on the client side. An example of a **Cookie-request** header is:

```
Set-Cookie: sessauth=44c46a10; expires=Wednesday, 27-Sep-2000
03:59:59 GMT
```

Here, the server requests that the client store the following cookie:

```
sessauth=44c46a10
```

Everything after the first semi-colon contains additional information about the cookie, such as the expiration date. When the e*Way sees this header, it extracts the cookie *sessauth=44c46a10* and returns it to the server on subsequent requests. The e*Way prepends a cookie header to the HTTP request, for example:

```
Cookie: sessauth=44c46a10
```

Each time the e*Way sends a request to the same server during a session, the cookie is sent along with the request.

1.3.5 SSL Support

The HTTP(S) e*Way offers complete support for the Secure Sockets Layer (SSL) security feature. See **Chapter 6** for details.

1.4 e*Way Java Classes

This section explains the Java classes in the HTTP(S) e*Way.

1.4.1 JDK Classes

The HTTP(S) e*Way primarily uses the **JDK 1.3 URL**, **URLConnection**, and **HttpURLConnection** classes. These classes allow for URL specification, opening and closing of connections, and the reading and writing of data through the use of abstracted streaming classes.

The **Authenticator** class is sub-classed in order to provide HTTP and proxy authentication. The **System** and **Security** classes are also used to support the setting of properties in order to use SSL.

1.4.2 SeeBeyondClasses

There is a total of ten classes that comprise the Java HTTP(S) e*Way. They are divided into the following groups:

- Core classes
  - **HttpAuthenticator**
  - **HttpClient**
  - **HttpClientAPI**
  - **HttpClientConnector**
  - **HttpHeader**
  - **HttpResult**

- SSL-related classes
  - **HTTPSSecurityProperties**
  - **HTTPSSystemProperties**
  - **SSLTunnelSocketFactory**
- Supporting classes
  - **QueryPair**
  - **QueryString**

1.4.3 Core Classes

The core classes implement the functionality of the e*Way by exposing the following objects and methods to the user:

- **HttpClientAPI** is the class that implements the HTTP functionality.
- **HttpClient** wraps the **HttpClient** API to enable the e*Gate Schema Designer to expose the **HttpClient** APIs via the **.xsc** Event Type Definition (ETD) file.
- **HttpAuthenticator** implements user-name and password authentication for both proxy authentication and/or HTTP Web site authentication.
- **HttpClientConnector** implements **EbobConnector** and is used to handle the e*Way's configuration.
- **HttpHeader** is used to encapsulate the name and value pair of an HTTP header (that is, "Accept-type" is the name and "text/xml" is the value).

Note: *Header names are case sensitive.*

The SSL-related classes are used to set up the JSSE run-time environment properties as follows:

- **HTTPSSecurityProperties** is used to set the security properties such as setting the provider information.
- **HTTPSSystemProperties** sets the system properties relating to security, such as the KeyStore file.
- **SSLTunnelSocketFactory** is used to create a tunneled socket connection to the Web server via a proxy.

The supporting classes are provided for convenience and further encapsulation as follows:

- **QueryPair** encapsulates the name and value pair for part of the query. Methods are exposed for setting both the name and values, without your having to specify the encoding. **QueryPair** handles the encoding for the user.
- **QueryString** represents the form data used to submit form data in general, whether it is a query or filling out some form data. **QueryString** uses **QueryPair** for the construction of the form data; essentially it is a collection of **QueryPair**.

1.5 e*Way Components

The following components comprise the Java-enabled HTTP(S) e*Way:

- **stchttp.jar** contains the logic required by the e*Way to gain access to the HTTP, and so on.
- **httpclient.xsc** allows the user to create hierarchical ETDs manually, to be used in conjunction with the parsing engine contained within the extended Java Collaboration Service.
- **e*Way Connection** provides the access to the information necessary for connection to a specified external connection.
- The Multi-Mode e*Way (a part of core e*Gate), including its executable file **stceway.exe**, provides a multi-threaded host for Java Collaborations.

A complete list of installed files appears in **Table 1 on page 18**.

1.6 Supported Operating Systems

The HTTP(S) e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP Tru64 V5.1A
- HP-UX 11.0 and 11i (PA-RISC)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0 and 11i (PA-RISC)
- Japanese Sun Solaris 8 and 9
- Korean Sun Solaris 8 and 9
- Traditional Chinese Windows 2000, Windows XP, and Windows Server 2003
- Traditional Chinese Sun Solaris 8 and 9

1.7 System Requirements

To use the HTTP(S) e*Way, you need to meet the following requirements:

- An e*Gate Participating Host
- A TCP/IP network connection

The e*Way must be configured and administered using the e*Gate Schema Designer.

Note: *Additional disk space can be required to process and queue the data that this e*Way processes. The amount necessary can vary based on the type and size of the data being processed and any external applications doing the processing.*

External System Requirements

There are no external system requirements.

Chapter 2

Installation

This chapter describes how to install the HTTP(S) e*Way. Although both the Monk and Java-enabled versions are installed, this document only addresses the Java-enabled version.

2.1 Windows Systems

2.1.1 Pre-installation

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

2.1.2 Installation Procedure

To install the HTTP(S) e*Way on Windows systems

1. Log in as an Administrator on the workstation on which 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 Autorun feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use Windows Explorer to launch the file **setup.exe** on the CD-ROM drive.
4. The **InstallShield** setup application launches. Follow the on-screen instructions to install the e*Way.

Note: *Be sure to install the e*Way files in the suggested **client** installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

2.2 UNIX Systems

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under the user name that 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 HTTP(S) e*Way on a UNIX system:

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

   cd /cdrom
4. Start the installation script by typing

   setup.sh
5. A menu of options appears. Select the "install e*Way" option. Then follow any additional on-screen directions.

Note: *Be sure to install the e*Way files in the suggested* ***client*** *installation directory. The installation utility detects and suggests the appropriate installation directory.* ***Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

2.3 Files/Directories Created by Installation

The HTTP(S) e*Way installation process installs the following files within the e*Gate directory tree. Files are installed within the **eGate** tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files and Directories Created by Installation

Directories	Files
client\classes\	stchttp.jar stcutil.jar
etd\httpclient\	httpclient.xsc
configs\httpclient\	httpclient.def
client\pkicerts\client	certmap.txt
client\pkicerts\trustedcas	GTECyberTrustGlobalRoot.cer MircrosoftRootAuthority.cer SecureServerCertificationAuthority.cer ThawtePremiumServerCA.cer ThawteServerCA.cer versisign_class3.cer
client\pkicerts\trustedstore	trustcacertsjks

Note: *See the **HTTP(S) e*Way Intelligent Adapter User's Guide (Monk Version)** for information on the files installed by the Monk-enabled version of the e*Way.*

Chapter 3

e*Way Connection Configuration

This chapter explains how to configure the HTTP(S) e*Way Connection.

3.1 Configuring e*Way Connections

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

To create and configure e*Way Connections

1. In the Schema Designer'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 this example, the connection has been defined as **SimpleHttpCP**.

   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 the connection type. For this e*Way, select **HTTP(S).**
7. Enter 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 Configuration Editor GUI.

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

9. Use the e*Way Configuration 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 Configuration Editor and save the new configuration file.

The HTTP(S) e*Way Connection configuration parameters are organized into the following sections:

3.1.1 HTTP Configuration

This section contains a set of top level parameters used by HTTP:

- **DefaultUrl**
- **AllowCookies**
- **ContentType**
- **AcceptType**

DefaultUrl

Description

Specifies the default URL to be used. If "https" protocol is specified, SSL must be configured. See **"SSL Configuration" on page 24**.

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

If using GET functionality, you can provide the parameters, using encoded query string notation. For example (all on one line):

http://www.ee.cornell.edu/cgi-bin/cgiwrap/~wes/pq?FirstName=John&LastName=Doe

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 set to a**pplication/x-www-form-urlencoded**. If you are sending other forms of data, set to the appropriate content-type. For example, **text/html**.

Accept-type

Description

Specifies the parameters for the **Accept-type** request header.

Required Values

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

3.1.2 Proxies Configuration

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

UseProxy

Description

Specifies whether an HTTP or HTTP(S) proxy is to be used. If it is set to HTTP, an HTTP proxy for non-secured connection is used. If HTTPS is selected, an HTTP(S) proxy for secured connection is used. Select **NO** if a proxy is not used.

Select **AUTO** to allow the e*Way to automatically switch between HTTP(S) proxy, when using a proxy for HTTP(S) connections, or HTTP proxy when using a proxy for HTTP connections. When **AUTO** is selected, both HTTP(S) and HTTP proxy hosts and ports must be specified. **AUTO** allows the e*Way to switch from an HTTP(S) URL to an HTTP URL and vice versa, via the proxy server.

See the configuration parameters **HttpProxyHost, HttpProxyPort, HttpsProxyHost, HttpsProxyPort, UserName**, and **Password** in this section for proxy usage.

Required Values

HTTP, **HTTPS**, or **NO**.

HttpProxyHost

Description

Allows you to specify either:

- The HTTP proxy host name where you can delegate requests to an HTTP server
- The reception of data from an HTTP server that can be delegated to a proxy

This parameter sets the proxy host for non-secured HTTP connections. To turn on proxy use, see the **UseProxy** configuration parameter.

Required Values

An 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 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 HTTP(S) 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 HTTP(S) proxy host number.

HttpsProxyPort

Description

Specifies the HTTP(S) 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 HTTP(S) proxy port name.

User Name

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 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** parameter.*

3.1.3 HttpAuthentication

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

UseHttpAuthentication

Description

Specifies whether standard HTTP Authentication is to be used. This parameter 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 the **UseHttpAuthentication** configuration parameter.

Required Values

A valid password.

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

3.1.4 SSL Configuration

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 HTTP(S) protocol. If set to **Yes**, then at least **HttpsProtocolImpl** and **Provider** must be given, as well as a valid **TrustStore** setting.

Required Values

Yes or **No**.

HttpsProtocolImpl

Description

Specifies the package that contains the HTTP(S) protocol implementation. This parameter 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 is the package that contains the SUN reference implementation of the HTTP(S) **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 parameter adds a JSSE provider implementation to the list of provider implementations. The default value is the SUN 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, you have to 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, you have to 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, you have to 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, 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.

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 key manager algorithm name to use. For example, the default key manager algorithm used in the SUN reference implementation of JSSE is **SunX509**.

Required Values

A valid key manager algorithm name.

TrustManagerAlgorithm

Description

Specifies the default trust manager algorithm name to use. For example, the default trust manager algorithm used in the SUN reference implementation of JSSE is **SunX509**.

Required Values

A valid trust manager algorithm name.

3.1.5 Connector Configuration

This section contains a set of top level parameters:

- **type**
- **class**
- **Property.Tag**

Type

Description

Specifies the type of connection.

Required Values

Http. The value defaults to HTTP; accept the default.

Class

Description

Specifies the class name of the HTTP Client connector object.

Required Values

A valid package name. The default is **com.stc.eways.http.HttpClientConnector**.

Property.Tag

Description

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

Required Values

A valid data source package name. Accept the default.

Chapter 4

Multi-Mode e*Way Configuration

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

4.1 Multi-Mode e*Way Properties

Set the Multi-Mode e*Way properties using the e*Gate Schema Designer.

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

1. Select the Navigator pane's Components tab in the Main window of the Schema Designer.
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 Configuration Editor window opens.
8. When the **Settings** page opens, set the configuration parameters for this e*Way's configuration file (see **"JVM Settings" on page 29** and **"General Settings" on page 31** for details).
9. After selecting the desired parameters, click **Save** on the **File** menu to save the configuration (**.cfg**) file.
10. Close the **.cfg** file and e*Way Configuration Editor.
11. Set the properties for the e*Way in the **e*Way Properties** dialog box.
12. Click **OK** to close the dialog box and save the properties.

4.2 JVM Settings

To correctly configure the HTTP(S) e*Way Intelligent Adapter, you must configure the Java Virtual Machine (JVM) settings. This section explains the configuration parameters in the e*Way Configuration Editor window, which control these settings.

JNI DLL Absolute Pathname

Description

Specifies the absolute pathname to where the JNI DLL installed by the *Java 2 SDK 1.2.2* is located on the Participating Host. This parameter is **mandatory**.

Required Values

A valid pathname.

Additional Information

The JNI dll name varies on different platforms as follows:

OS	Java 2 JNI DLL Name
Windows systems	jvm.dll
Solaris 2.6, 2.7, 2.8	libjvm.so
HP-UX	libjvm.sl
AIX 4.3	libjvm.a
HP Tru64	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 platforms.

To ensure that the JNI DLL 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 that contain shared libraries (UNIX) or DLLs (Windows).

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the Java VM.

Required Values

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

Additional Information

If this parameter is 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 Java Virtual Machine (VM). This parameter is optional. If it is eft unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) is set.

Note: *All necessary JAR and ZIP files needed by both e*Gate and the Java VM 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%
```

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 Java VM is 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 Java VM is 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 is 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 Java VM is used.

Required Values

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

Disable JIT

Description

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

Required Values

YES or **NO**.

Note: *This parameter is not supported for Java Release 1.*

Allow Remote Debugging of JVM

Description

Specifies whether to allow remote debugging of the JVM.

Required Values

YES or **NO**.

4.3 General Settings

This section contains the parameters for rollback wait and IQ messaging priority.

Note: *For more information on the **General Settings** configuration parameters see the **e*Gate Integrator User's Guide**.*

4.3.1 Rollback Wait Interval

Description

Specifies the time interval to wait before rolling back the transaction.

Required Values

A number within the range of 0 to 99999999, representing the time interval in milliseconds.

4.3.2 Standard IQ FIFO

Description

Specifies whether the highest priority messages from all SeeBeyond Standard IQs are delivered in the first-in-first-out (FIFO) order.

Required Values

Select **Yes** or **No**. **Yes** indicates that the e*Way retrieves messages from all SeeBeyond Standard IQs in the first-in-first-out (FIFO) order. **No** indicates that this feature is disabled; **No** is the default.

Chapter 5

Implementation

This chapter explains how to implement the HTTP(S) e*Way Intelligent Adapter in a production environment, using a sample e*Gate Integrator schema.

5.1 Implementation Overview

This section explains how to implement the HTTP(S) e*Way using e*Gate Integrator schema sample included on your installation CD-ROM. Find this sample on the CD-ROM at the following path location:

\samples\ewhttp

This sample allows you to observe and/or create end-to-end data-exchange scenarios involving e*Gate, the e*Way, and HTTP. This chapter explains how to implement the the sample schemas for the e*Way.

Schema Example

In addition, the chapter contains an example of a schema implementation, to give you additional practice and knowledge about implementing the HTTP(S) e*Way. You must create this schema from scratch, after you have installed the e*Way.

See **"Implementing a Schema Example" on page 41** for the complete steps that explain how to create this schema.

5.2 Implementing the Sample Schema

This section explains how to implement the sample schema for the HTTP(S) e*Way. The schema demonstrates how to configure the essential features of the e*Way in a typical e*Gate environment.

5.2.1 JavaHttpSample Sample Schema: Overview

This section provides an overview of how to configure the sample schema and how the schema operates. The name of this schema is JavaHttpSample, and it is contained in the schema import file **JavaHttpSample.zip**.

This section also explains how to use the HTTP(S) e*Way within a sample schema. The sample sends and receives Events from any platform supported by the e*Way.

It is assumed that the e*Way has been installed properly and that all the necessary files and scripts are in the default directory locations. You must also have access to a remote Web site via HTTP.

Note: *For more information, see the* ***Readme.txt*** *file that accompanies the sample.*

Before Schema Implementation

To use and implement a sample schema, the HTTP(S) e*Way and sample schema must be installed, all of the necessary files and scripts must be located in the default location, and you must also have access to a remote Web site via HTTP.

Schema Operation

The schema operates as follows:

- Posting to a Web site:
  - Accepts an input file with data from a Web server
  - Posts the data to a remote Web site via HTTP
- Getting a Web page:
  - Receives a page from a remote Web site
  - Outputs the page to a local file system

The input message Event is an Extensible Markup Language (XML) input file that enters the schema via an inbound file e*Way. The input XML message **HttpRequest** has the following child tags:

- **Method**: Denotes **POST** or **GET**.
- **URL**: Specifies the Web site to connect to.
- **Data**: Contains form data or raw data for the **POST** method.

The following examples illustrate the four possible ways to populate the XML input message Event for an HTTP request:

- Posting to a Web site (using **POST**) with form data (each name/value pair is delimited by | with ^ used to separate the name part from the value part) as follows:

```
<HttpRequest>
<Method>POST</Method>
<URL>https://hypnos.stc.com:444/access-controlled/query-
response.pl</URL>
<Data>cp^Cool|version^5.0.1</Data>
</HttpRequest>
```

- Posting to a Web site (using **POST**) with raw data as follows:

```
<HttpRequest>
<Method>POST</Method>
<URL>http://myhost.seebeyond.com/examples/servlet/SampleServlet<
    URL>
<Data>This is a sample text to post to my servlet.</Data>
</HttpRequest>
```

- Getting a Web page (using **GET**; there is no data in the **Data** tag):

```
<HttpRequest>
<Method>GET</Method>
<URL>http://www.seebeyond.com</URL>
<Data></Data>
</HttpRequest>
```

- Getting a Web page (using **GET**) with URL-encoded name/value pairs (there is no data in the **Data** tag) as follows:

```
<HttpRequest>
<Method>GET</Method>
<URL>https://myhost.seebeyond.com/examples/servlet/
RequestParamExample?firstname=John&lastname=Doe</URL>
<Data></Data>
</HttpRequest>
```

Note: *The Web page must support either the **POST** or **GET** method, or both.*

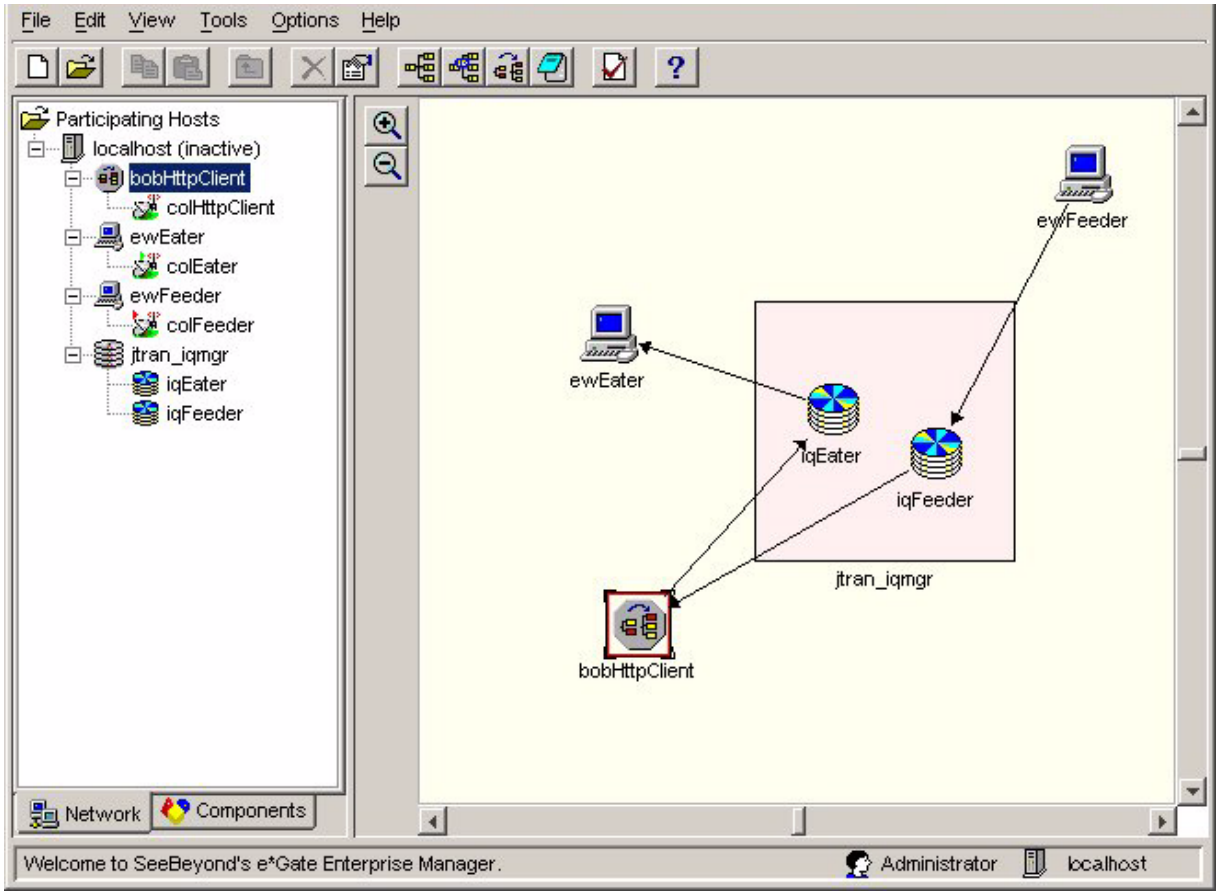
This XML Event is the trigger to the Java Collaboration. The **executeBusinessRule** method then populates the **HttpClient** Event, using the data supplied by the XML Event. The Collaboration then executes the **POST** or **GET** method as specified by the **Method** tag.

The Java Collaboration then checks for an **HTTP_OK** status code and gets the response from the Web server. The response is placed in a "blob" that is routed to an IQ to be picked up by the outbound e*Way.

Schema Setup

Figure 1 on page 36 shows the schema's general architecture, using the e*Gate Schema Designer's Network View.

Figure 1 JavaHttpSample Schema in Network View



Schema Components

This sample schema implementation consists of the following basic components:

- **ewFeeder:** Inbound file e*Way receives XML file input from an external source.
- **colFeeder:** Collaboration for the **ewFeeder** e*Way, receives the external files, applies pass-through Collaboration Rules, and publishes the information to the **iq_Feeder** IQ.
- **ewEater:** Outbound file e*Way sends information to an external system.
- **colEater:** Collaboration for the **ewEater** e*Way receives information from **bobHttpClient** via the **iq_Eater** IQ and publishes it to an external system.
- **bobHttpClient**: Business Object Broker (BOB) component holds the **colHttpClient** Collaboration.

- **colHttpClient**: Collaboration applies extended Java Collaboration Rules to inbound Events to perform the desired business logic, then publishes the Events to the **iq_Eater** IQ.
- **iq_Feeder**: Intelligent Queue (IQ) is a SeeBeyond Standard IQ that receives data from **ewFeeder** and sends data to the **bobHttpClient** BOB.
- **iq_Eater**: IQ is a SeeBeyond Standard IQ that receives data from **bobHttpClient** and sends data to the **ewEater**.
- **JavaHttpClient**: e*Way Connection that receives the **JavaHttpClient** Event from **colHttpClient** (BOB Collaboration) and executes the **POST** and/or **GET** methods to the remote Web site.

Table 2 lists all the components for the schema, along with their logical names and configuration settings.

Table 2 JavaHttpSample Schema Components

Component	Logical Name	Settings
Schema	JavaHttpSample	
Control Broker	localhost_cb	
IQ Manager	jtran_iqmgr	Start Up = Auto
Event Types	JavaHttpClient	
	JavaHttpRequest	
	ResultAsBlob	
Java ETDs	httpclient.xsc	▪ Package Name = HttpClientPackage
	JavaHttpRequest.xsc	▪ Package Name = JavaHttpRequestPackage
	ResultAsBlob.xsc	▪ Package Name = ResultAsBlobPackage
Collaboration Rules	FeederPassThru	▪ Service = Pass Through ▪ Subscription = JavaHttpRequest ▪ Publication = JavaHttpRequest
	EaterPassThru	▪ Service = Pass Through ▪ Subscription = ResultAsBlob ▪ Publication = ResultAsBlob
	ExecuteJavaHttp Request	▪ Service = Java ▪ Input = ♦ HttpRequest; JavaHttpRequest.xsc (Trigger) ▪ Output = ♦ HttpResult; ResultAsBlob.xsc (Trigger N/A) ♦ HttpClient; httpclient.xsc (Trigger N/A)

Table 2 JavaHttpSample Schema Components (Continued)

Component	Logical Name	Settings
Collaborations	colFeeder	▪ Collaboration Rule = FeederPassThru ▪ Subscription = JavaHttpRequest Event from <EXTERNAL> ▪ Publication = JavaHttpRequest Event to iq_Feeder IQ
	colEater	▪ Collaboration Rule = EaterPassThru ▪ Subscription = ResultAsBlob Event from colHttpClient BOB Collaboration ▪ Publication = ResultAsBlob Event to <EXTERNAL>
	colHttpClient	▪ Collaboration Rule = ExecuteJavaHttp Request ▪ Subscription = JavaHttpRequest Event from colFeeder e*Way Collaboration ▪ Publication = ♦ JavaHttpClient to JavaHttpClient e*Way Connection ♦ ResultAsBlob to iq_Eater IQ
e*Way Connection	JavaHttpClient	▪ 100 for Event Type "get" interval
IQs	iq_Feeder	▪ 100 for Event Type "get" interval
	iq_Eater	▪ 100 for Event Type "get" interval
Inbound e*Way	ewFeeder	▪ Executable = stcewfile.exe ▪ Configuration file = ewFeeder.cfg ▪ Start Up = Auto ▪ Collaboration = colFeeder
Outbound e*Way	ewEater	▪ Executable = stcewfile.exe ▪ Configuration file = ewEater.cfg ▪ Start Up = Auto ▪ Collaboration = colEater
Business Object Broker (BOB)	bobHttpClient	▪ Executable = stcbob.exe ▪ Start Up = Auto ▪ Collaboration = colHttpClient

Additional Information

SSL Configuration

The e*Way Connection **JavaHttpClient** in this schema is configured to use the Secure Sockets Layer (SSL) feature. To enable SSL in the e*Way, select **Yes** under the **UseSSL** parameter in the e*Way's **SSL Configuration** section (see **"SSL Configuration" on page 24**).

Note: *To use clear HTTP instead of HTTPS, select* ***No****.*

Configure the appropriate SSL parameters in the e*Way Connection as follows:

- **KeyStore**: Specify the full path and file name for the e*Way's KeyStore. The KeyStore contains the e*Way's private key and Certificate Authority (CA) certificate.
- **KeyStoreType**: Specify the KeyStore type being used (either JKS or PKCS12).
- **KeyStorePassword**: Specify the password used to access the KeyStore. This password is created when the KeyStore is created, using either KeyTool for JKS or **openssl** for PKCS12. Be sure to press ENTER after entering the password.
- **TrustStore**: Specify the full path and file name for the e*Way's TrustStore. The TrustStore contains trusted CA certificates.
- **TrustStoreType**: Specify the TrustStore type being used, currently only JKS.
- **TrustStorePassword**: Specify the password used to access the TrustStore. This password is created when the TrustStore is created, with the first import of a CA certificate. Be sure to press ENTER after entering the password.

Note: *For more information on SSL and how to use it with this e*Way, see* **Chapter 6**.

User Name and Password Authentication

In the sample schema, **UseHttpAuthentication** has been selected with the user name and password given in the **HttpAuthentication** section of the e*Way Connection. If you are accessing a Web site that requires user name and password authentication, be sure to supply the correct user name and password. Otherwise, you do not have to enter these parameters.

Using Proxies

By default, the **UseProxy** parameter (under the **Proxies** section of the e*Way Connection configuration) is selected as **AUTO**. To disable proxy usage, select **No**. See the chapter details on this parameter.

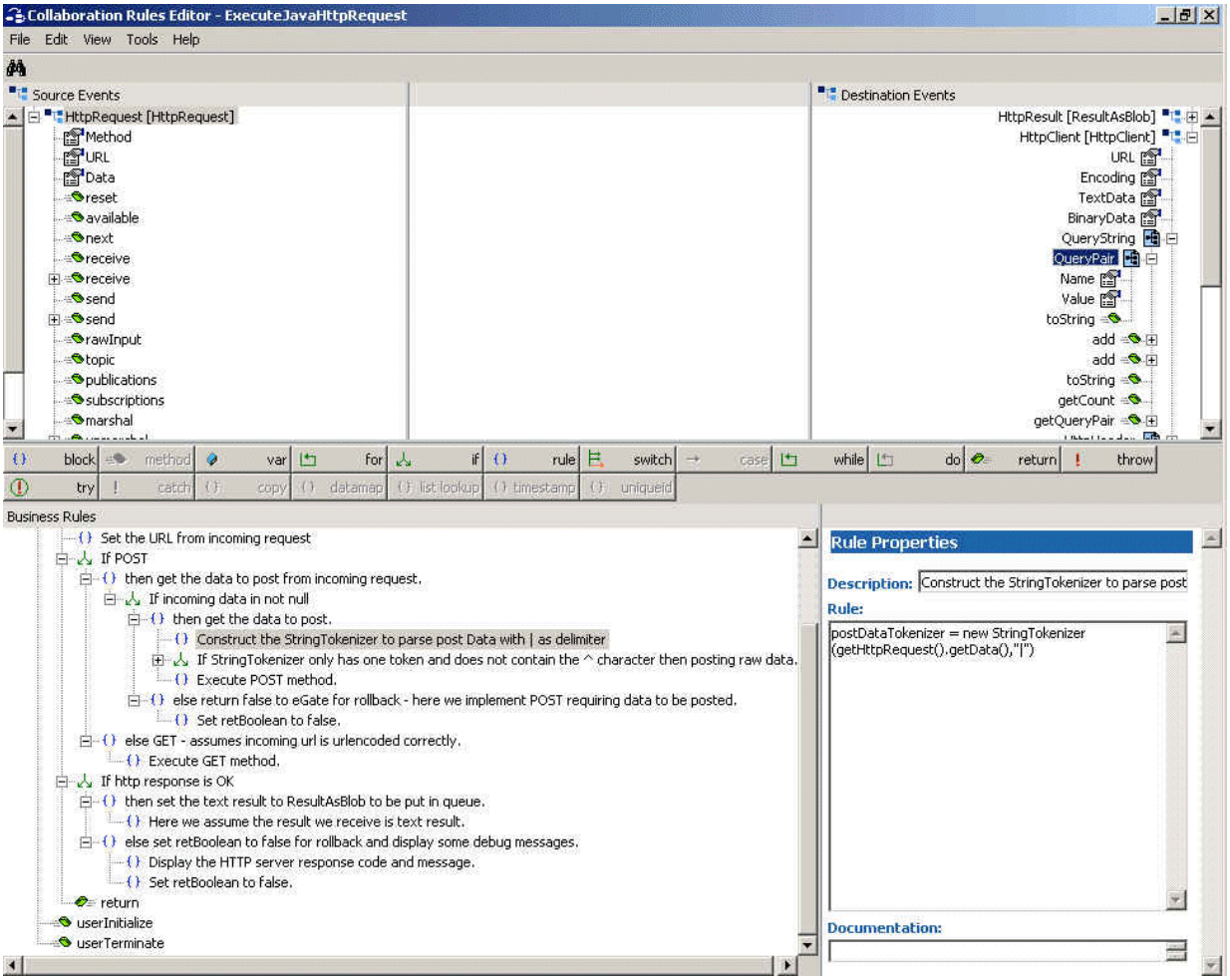
Note: *The **AUTO** parameter allows the e*Way to use the proxy for both HTTP and HTTP(S) URLs.*

See **Chapter 3** for more details on the e*Way's configuration parameters.

Posting Form Data

You can post form data using the **QueryString** node of the **httpclient.xsc** ETD. In this ETD, you can cut and paste rules that add a name/value pair before posting. Figure 2 shows the e*Gate Collaboration Rules Main window and the **ExecuteJavaHttpRequest** Collaboration Rule.

Figure 2 Collaboration Rules Editor Main Window: httpclient.xsc QueryString Node



For more information, open the **ExecuteJavaHttpRequest** Collaboration Rule shown in Figure 2, using the Collaboration Rules Editor, and observe how the sample Business Rules are constructed.

Running a Schema

To run a sample schema

- From the command line prompt, enter on a single line:

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

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

The schema components start automatically. When there are no more run-time messages, check the remote Web site. If the subject data is posted to the desired Web page, the schema operation has been successful.

Caution: *While the schema is running, opening the destination file can cause errors.*

5.2.2 Importing a Schema

If desired, you can import the sample schema and view its features and configuration without creating it yourself.

To import a sample schema

1. Copy the desired **.zip** file, **JavaHttpSample.zip,** from the **\Samples\ewhttp** directory in the install CD-ROM to your desktop or to a temporary directory, then unzip the file.
2. Start the e*Gate Schema Designer.
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.
6. Click the **.zip** file then click **Open**.

   The sample schema is installed.

5.3 Implementing a Schema Example

This section explains how to create a schema that implements the HTTP(S) e*Way. With this example, you are creating your own implementation from scratch. The section leads you through step-by-step procedures that explain how to create and configure your own schema.

5.3.1 Implementing the e*Way: Overview

During installation, the host and Control Broker are automatically created and configured. The default name of each is the name of the host on which you are installing the e*Gate Schema Designer.

To complete the implementation of the e*Way, do the following operations:

- Make sure that the Control Broker is activated.
- Using the e*Gate Schema Designer, define and configure the following components as necessary:
  - Inbound e*Way using **stcewfile.exe**
  - Outbound e*Way using **stcewfile.exe**
  - The Multi-Mode e*Way component.
  - Event Types and Event Type Definitions (ETDs) used to package the data to be exchanged with the external system
  - Collaboration Rules to process ETDs
  - The e*Way Connection to be created (see **Chapter 3** for details)
  - Collaborations, to be associated with each e*Way component, to apply the required Collaboration Rules
  - The destination where data is published before being sent to the external system

The rest of this chapter explains how to define and associate each of the previously listed components, creating and configuring your own HTTP(S) e*Way schema.

Note: *For complete information on how to create and configure an e*Gate schema, see the* ***e*Gate Integrator User's Guide*** *and* ***Creating an End-to-end Scenario with e*Gate Integrator****.*

5.3.2 Before Implementation

Be sure that the following conditions have been met before you begin to create this example:

- The HTTP(S) e*Way has been successfully installed.
- The e*Way's executable and the configuration files have been appropriately assigned.
- All necessary .**jar** files are accessible to the e*Way.

The rest of this section explains how to create and implement the schema example.

5.3.3 Creating the New Schema

The first task in deploying the schema example is to create a new schema name. While it is possible to use the default schema for this example implementation, it is recommended that you create a separate schema for testing purposes.

After you install the HTTP(S) e*Way, do the following steps:

1. Start the e*Gate Schema Designer.
2. When the Schema Designer prompts you to log in, select the host that you specified during installation, and enter your password.

3 You are then be prompted to select a schema. Click on **New**.

4 Enter a name for the new schema. In this case, enter **HTTP_Test_New**, or any name as desired.

The Schema Designer opens under your new schema. You can access the ETD Editor and Collaboration Rules Editor features via the Schema Designer. You are now ready to begin creating the necessary components for this schema example.

5.3.4 Creating Event Types and ETDs

The HTTP(S) e*Way installation includes the file **httpclient.xsc**, which represents an HTTP(S) ETD template.

Creating an ETD from an Existing DTD

For the purpose of this example, the following procedure shows how to create an ETD as the input file, from a .**dtd** file. In this procedure, you create an Event Type and its corresponding ETD.

To create an ETD file from a .dtd file

1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Schema Designer's Main window.

2 On the palette, click the icon to create a new **Event Type**.

3 Enter the name of the **Event**, then click **OK.** For this sample, the first Event Type is defined as **HttpEvent**.

4 Select the new **Event Type**, then right-click to edit its properties.

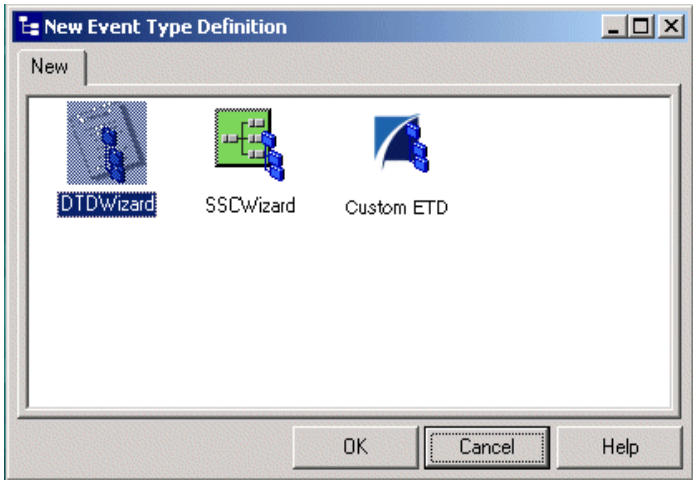
5 When the **Properties** window opens, click the **New** button.

The ETD Editor's Main window opens.

6 Select **New** from the **File** menu on **Task Manager**.

The **New Event Type Definition** dialog box appears (see Figure 3).

Figure 3 New Event Type Definition Dialog Box



7 Select the desired wizard icon. For this ETD, select **DTDWizard**.

Note: *For complete information on how to use the DTD wizard, see the* ***XML Toolkit*** *document.*

8 Enter a package name where the DTD Builder can place all the generated Java classes associated with the created ETD. For this sample use SimpleHTTP.

9 Select a DTD file to be used by the DTD Builder to generate an ETD file.

10 Using the **Browse** button, navigate to and select an existing DTD.

11 Click **Next**. and review the summary information.

12 Click **Back** to edit; otherwise, click **Finish**.

The ETD Editor displays a graphical illustration of the newly converted .**xsc** file.

13 Save the file and name it **HttpEvent.xsc**, then promote it to run time.

Creating an ETD Without an Existing DTD

For the purpose of this example, the following procedure shows how to create an ETD without using an existing .**dtd** file as the input file. In this procedure, you create and Event Type and its corresponding ETD.

To create an ETD file without a .dtd file

1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Schema Designer's Main window.

2 On the palette, click the icon to create a new **Event Type**.

3 Enter the name of the **Event**, then click **OK.** For this sample, the first Event Type is defined as **Outgoing_Event**.

4 Select the new **Event Type**, then right-click to edit its properties.

5 When the **Properties** dialog box opens, click the **New** button

The ETD Editor's Main window opens.

6 Select **New** from the **File** menu on **Task Manager**.

7 The **New Event Type Definition** dialog box appears (see **Figure 3 on page 44**).

8 Select the desired wizard icon. For this ETD, select **Custom ETD**.

9 Enter a package name where the DTD Builder can place all the generated Java classes associated with the created ETD. For this sample, use **SimpleHTTP** as the package name.

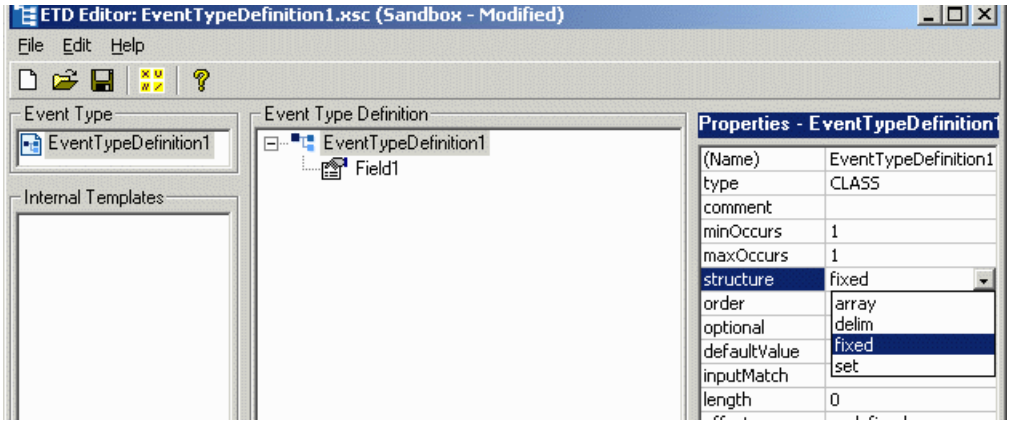
10 Return to the ETD Editor's Main window and select **New** from the **File** menu.

11 Select **EventTypeDefintion1**.

12 Right click, select **Add Field, as Child Node**.

13 Change the structure type to fixed (see Figure 4).

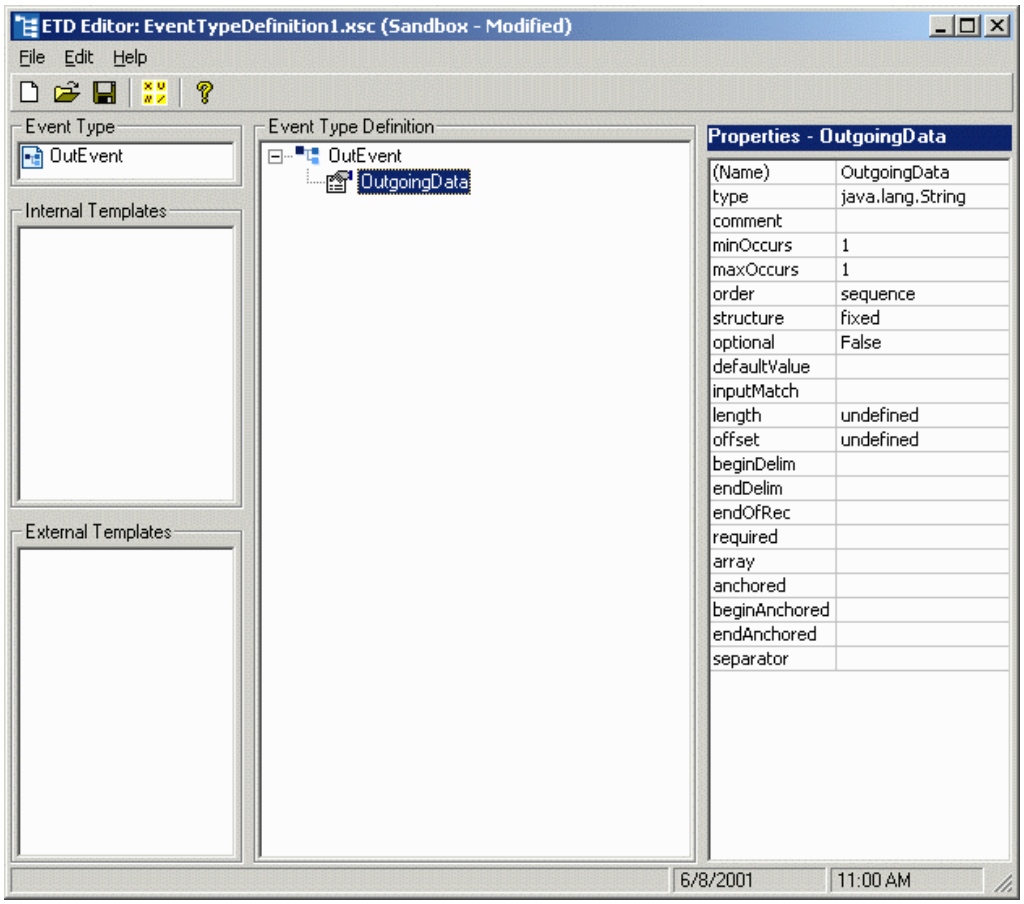
Figure 4 ETD Editor: Changing Structure Type



14 Right-click on the **EventTypeDefinition1** and rename it to **OutEvent**.

15 Right-click on the **Field1** and rename it to **OutgoingData** (see **Figure 5 on page 46**).

Figure 5 ETD Editor: Renaming ETD Elements



16 Save the file and name it **OutgoingEvent.xsc**, then promote it to run time.

Creating a New ETD from an Existing .xsc File

For the purpose of this example, the following procedure shows how to create a new ETD from an existing **.xsc** file using **HttpClient.xsc** as the input file. In this procedure, you create an Event Type and its corresponding ETD.

To create an ETD file from another .xsc file

1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Schema Designer's Main window.

2 On the palette, click the icon to create a new **Event Type**.

3 Enter the name of the **Event**, then click **OK.** For this sample, the first Event Type is defined as **HttpClient**.

4 Select the new **Event Type**, then right-click to edit its properties.

5 When the **Properties** dialog box opens, click the **Find** button.

6 Select **HttpClient.xsc** (provided as the default destination **.xsc** file).

7 Click **OK** to continue.

5.3.5 Creating and Configuring e*Ways

The first components to be created are the following e*Ways:

- **Inbound_eWay**
- **Outbound_eWay**
- **Multi-Mode_eWay**

This section provides instructions for creating each e*Way.

Inbound e*Way

To create the inbound e*Way

1. Select the Navigator's **Components** tab.
2. Open the host on which you want to create the e*Ways.
3. Select the **Control Broker** that manages the new e*Ways.
4. On the palette, click the e*Way icon.
5. Enter the name of the new e*Way (in this case, **Inbound_eWay)**, then click **OK**.
6. Select **Inbound_eWay**, then double-click to edit its properties.
7. When the **e*Way Properties** dialog box appears, click on the **Find** button beneath the **Executable File** field, and select **stcewfile.exe** for the executable file.
8. Under the **Configuration File** field, click on the **New** button. When the **Settings** configurations of the e*Way Configuration Editor appear, set the parameters for **Settings** as shown in Table 3.

Table 3 Configuration Parameters for Inbound e*Way

Parameter	Value
General Settings	
AllowIncoming	Yes
AllowOutgoing	No
Outbound Settings	Default
Poller Inbound Settings	
PollDirectory	C:\Indata (input file folder)
InputFileExtension	*.fin (input file extension)
PollMilliseconds	Default
Remove EOL	Default
MultipleRecordsPerFile	Default
MaxBytesPerLine	Default
BytesPerLineIsFixed	Default

9. Save the **.cfg** file, and exit from **Settings**.

10 After selecting the desired parameters, save the configuration file and promote the file to run time. Close e*Way Configuration Editor and the **.cfg** file.

11 Use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each e*Way you configure as follows:

   A Use the **Startup** tab to specify whether the e*Way starts automatically, or restarts after abnormal termination or due to scheduling, and so on.

   B Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

   C Use **Security** to view or set privilege assignments.

12 Click **OK** to close the **e*Way Properties** dialog box.

Outbound e*Way

To create the outbound e*Way

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Ways.

3 Select the **Control Broker** that manages the new e*Ways.

4 On the palette, click the e*Way icon.

5 Enter the name of the new e*Way (in this case, **Outbound_eWay)**, then click **OK**.

6 Select **Outbound_eWay**, then double-click to edit its properties.

7 When the **e*Way Properties** dialog box appears, click the **Find** button beneath the **Executable File** field, and select **stcewfile.exe** for the executable file.

8 Under the **Configuration File** field, click on the **New** button. When the **Settings** configurations of the e*Way Configuration Editor appear, set the parameters for **Settings** as shown in Table 4.

Table 4 Configuration Parameters for Outbound e*Way

Parameter	Value
General Settings	
AllowIncoming	No
AllowOutgoing	Yes
Outbound Settings	
OutputDirectory	C:\DATA\HTTP
OutputFileName	output%d.dat
MultipleRecordsPerFile	Yes
MaxRecordsPerFile	10000
AddEOL	Yes
Poller Inbound Settings	Default
Performance Testing	Default

9 Exit from **Settings**.

10 After selecting the desired parameters, save the configuration file and promote the file to run time. Close e*Way Configuration Editor and the **.cfg** file.

Multi-Mode e*Way

To create the Multi-Mode e*Way

1 Select the **Navigator's Components** tab.

2 Open the host on which you want to create the e*Way.

3 Select the **Control Broker** that manages the new e*Way.

4 On the palette, click the icon to create a new **e*Way**.

5 Enter the name of the new e*Way (**Http_Multi_Mode**), then click **OK**.

6 Select the new component, then double-click to edit its properties.

7 When the **e*Way Properties** dialog box appears, use the default executable file, **stceway.exe**.

8 To edit the JVM Settings, select **New** under Configuration file.

See **"Multi-Mode e*Way Configuration" on page 28** for details on the parameters associated with the Multi-Mode e*Way.

9 Save the **.cfg** file, and exit from **Settings**.

10 Use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each.

   A Use the **Startup** tab to specify whether the Multi-Mode e*Way starts automatically, restarts after abnormal termination or due to scheduling, etc.

   B Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

   C Use **Security** to view or set privilege assignments.

11 Select **OK** to close the **e*Way Properties** dialog box.

5.3.6 Creating the e*Way Connection

The e*Way Connection configuration file contains the connection information, along with the information needed to communicate via HTTP(S).

To create and configure a New e*Way Connection

1 Highlight the **e*Way Connection** folder on the **Components** tab of the e*Gate Navigation pane.

2 On the palette, click the icon to create a new e*Way Connection.

3 Enter the name of the e*Way Connection, then click **OK.** For the purpose of this example, the first Event Type is defined as **HttpEP**.

4 Select the new **e*Way Connection**, then right-click to edit its properties.

5 When the **Properties** window opens, select HTTP/HTTP(S) from the **e*Way Connection Type** drop-down menu.

6 Under **e*Way Connection Configuration File**, click **New**.

7 After the e*Way Configuration Editor opens, select the desired parameters.

   For more information on the e*Way's **Connection Type** parameters, see **Chapter 3**.

8 Save the **.cfg** file and promote it to run time.

5.3.7 Creating IQs

The next step is to create and associate an IQ. These components manage the exchange of information 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, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

To create and modify an IQ for the HTTP(S) e*Way

1 Select the Navigation pane's **Components** tab.

2 Open the host on which you want to create the IQ.

3 Open a **Control Broker**.

4 Select an **IQ Manager**.

5 On the palette, click the IQ icon.

6 Enter the name of the new IQ (**iq_standard**), then click **OK**.

7 Select the new IQ, then double-click to edit its properties.

The **IQ Properties** dialog box appears.

8 On the **General** tab, specify the **Service** and the **Event Type Get Interval**.

The **Standard_STC** IQ Service provides sufficient functionality for most applications. If specialized services are required, custom IQ Service .**dll** files can be created.

Use the default **Event Type Get Interval** of 100 ms for this implementation.

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

10 Close the **IQ Properties** dialog box.

5.3.8 Creating Collaboration Rules

The next step is to create the Collaboration Rules that extract and process selected information from the source Event Type defined previously, according to its associated Collaboration Service. The **Default Editor** can be set to either the Monk or Java programming language.

From the **Schema Designer Task Bar,** select **Options** and click **Default Editor**. For this example, set the default to Java.

To create a Collaboration Rules file

1 Select the Navigation pane's **Components** tab in the e*Gate Schema Designer.

2 In the Navigation pane, select the **Collaboration Rules** folder.

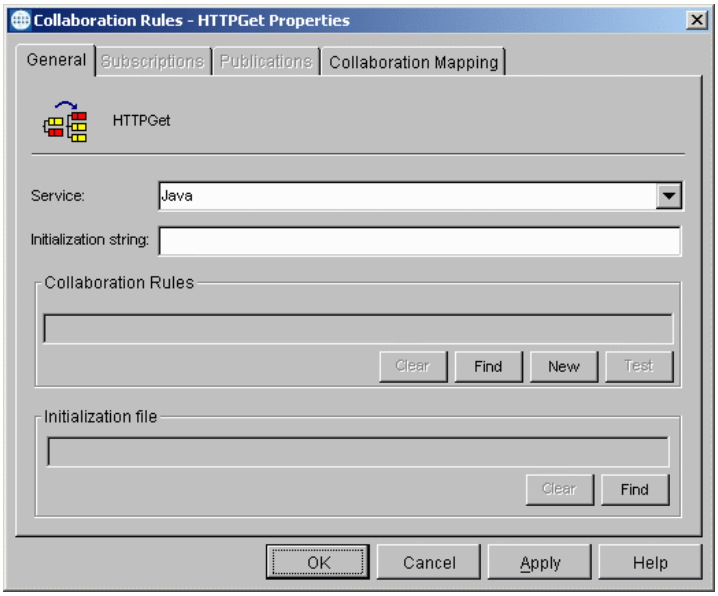
3 On the palette, click the Collaboration Rules icon.

4 Enter the name of the new Collaboration Rule, then click **OK. Http_Get** is used for this example.

5 Select the new **Collaboration Rule**, then right-click to edit its properties.

The **Collaboration Rules Properties** dialog box appears (see Figure 6).

Figure 6 Collaboration Rules Properties Dialog Box

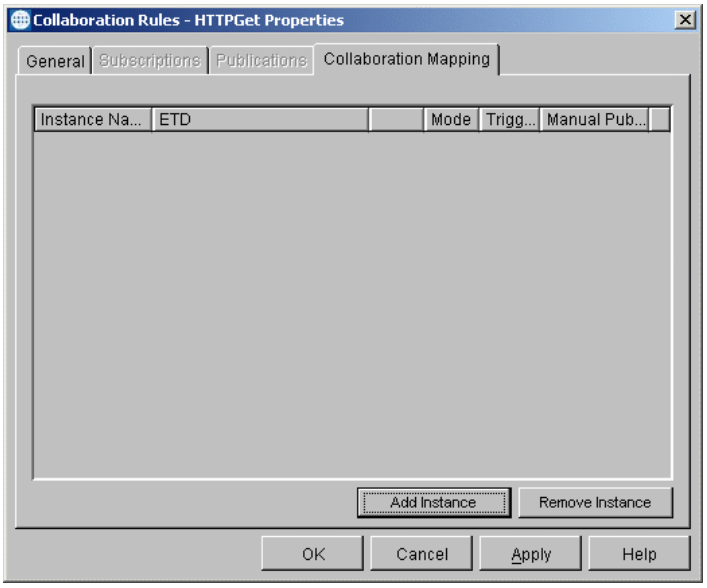


6 On the **General** tab in the dialog box select the **Java Collaboration Service**. **HTTPGet** uses the e*Gate Java Collaboration Service to manipulate Events and Event data.

7 In the **Initialization string** box, enter any required initialization string that the Collaboration Service may require. This field can be left blank.

8 Click the **Collaboration Mapping** tab (see Figure 7).

Figure 7 Collaboration Rules Properties Collaboration Mapping Tab



9 Using the **Add Instance** button, create instances to coincide with the Event Types.

10 In the **Instance Name** column, enter **In** for the instance name.

11 Click **Find**, navigate to **etd\HttpEvent.xsc**, double-click to select.

HttpEvent.xsc is added to the ETD column of the instance row.

12 In the Mode column, select **In** from the drop–down menu available.

13 In the Trigger column, click the box to enable trigger mechanism.

14 Repeat steps 9 through 13 using the following values:

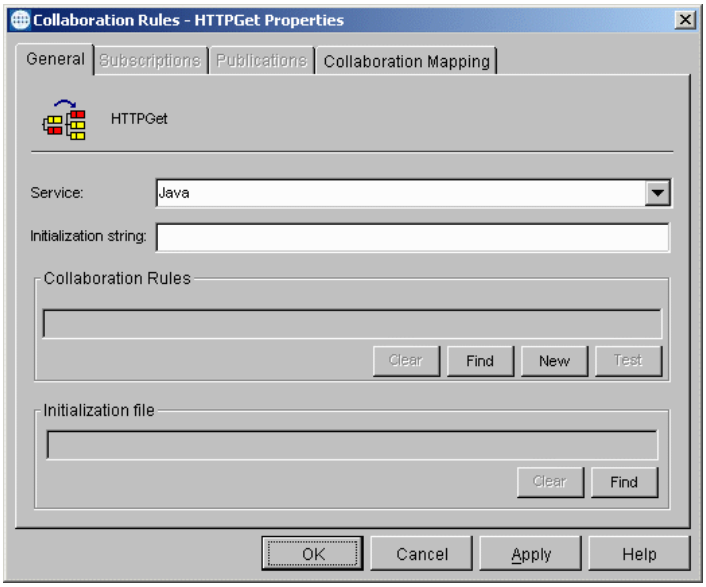
- Instance Name: **Out**
- ETD: **Outgoing_Event.xsc**
- Mode: **Out**
- Trigger: Do not select.

15 Repeat steps 9 through 13 using the following values:

- Instance Name: **HttpClient**
- ETD: **HttpClient.xsc**
- Mode: **Out**
- Trigger: Do not select.

16 Select the **General** tab again, then click **New** (see Figure 8).

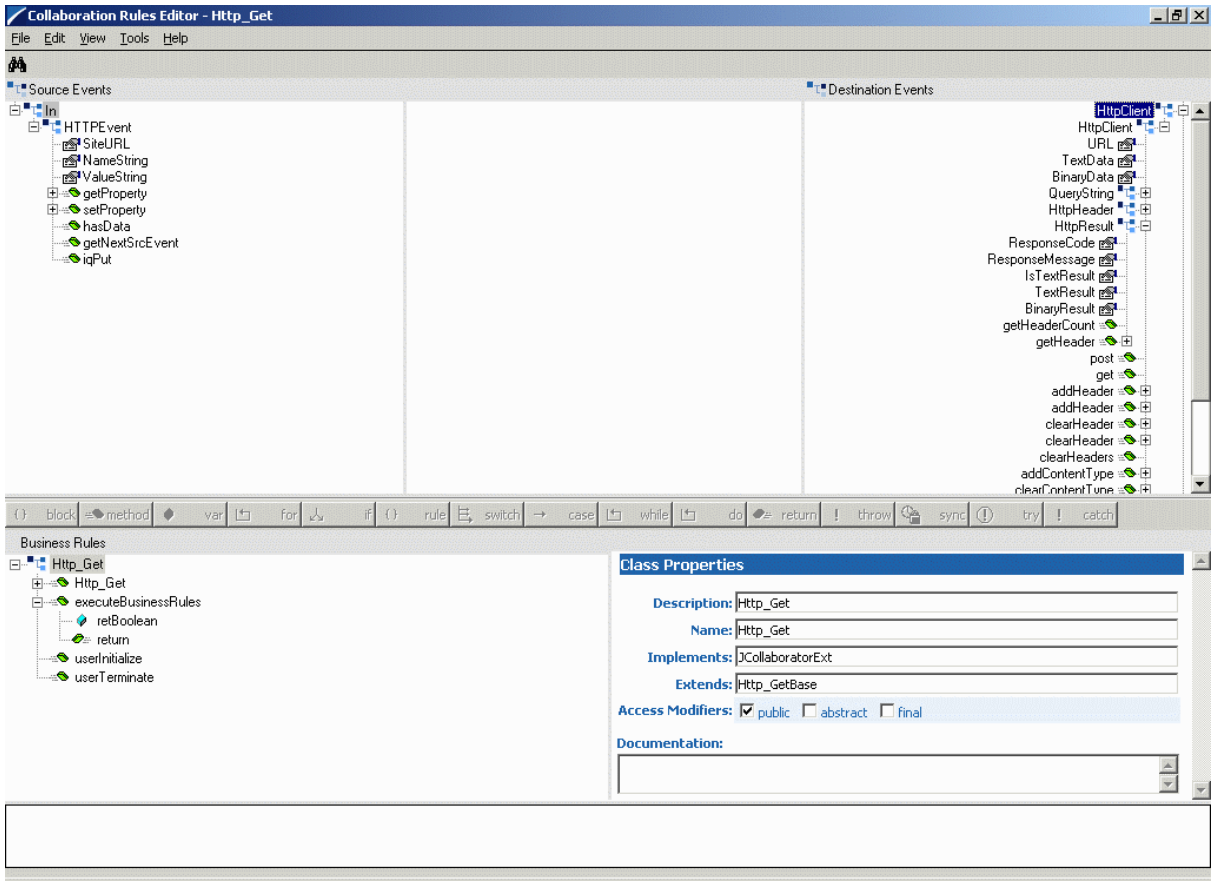
Figure 8 Collaboration Rules Properties General Tab



The Collaboration Rules Editor Main window opens.

17 Expand the window to full size for optimum viewing, expanding the source and destination Events also (see **Figure 9 on page 54**).

Figure 9 Collaboration Rules Main Window: Mapping and Properties



You are now ready to go on to the next procedure, creating the Collaboration Rules class.

To create the Collaboration Rules class

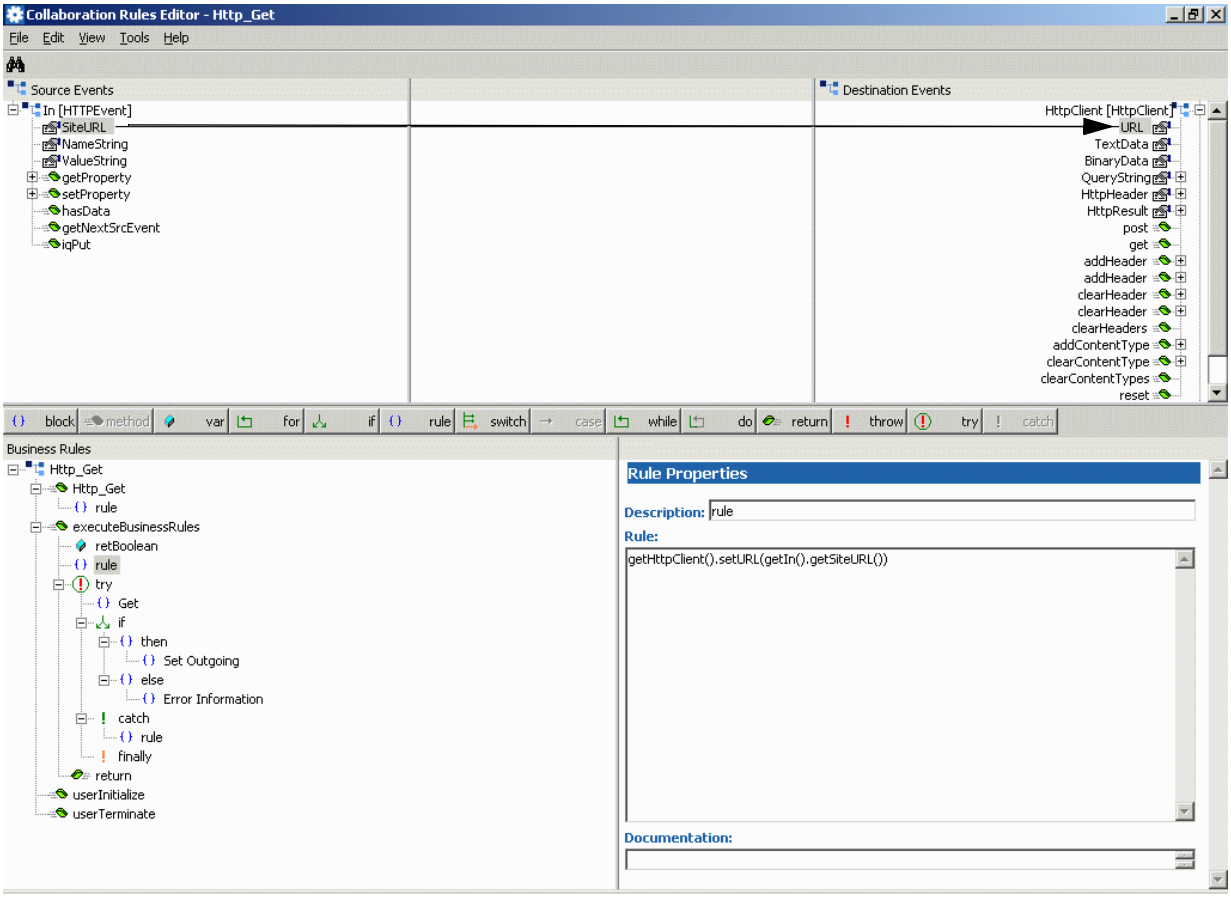
1 Highlight **retBoolean** in the **Business Rules** pane.

All of the user–defined business rules are added as part of this method.

2 Select **SiteUrl** from the **Source Events** pane. Drag and drop it onto **URL** in the **Destination Events** pane.

A connecting line appears between the properties objects. In the **Business Rules** pane, a rule expression appears, with the properties of that rule displayed in the Rule Properties pane (see Figure 10).

Figure 10 Collaboration Rules Main Window: After Drag and Drop



3 Select the newly created rule highlighted in Figure 10.

4 Change the description of the method from **rule** to **UrlMapping**. Reselect the **rule** to affect the updated description name.

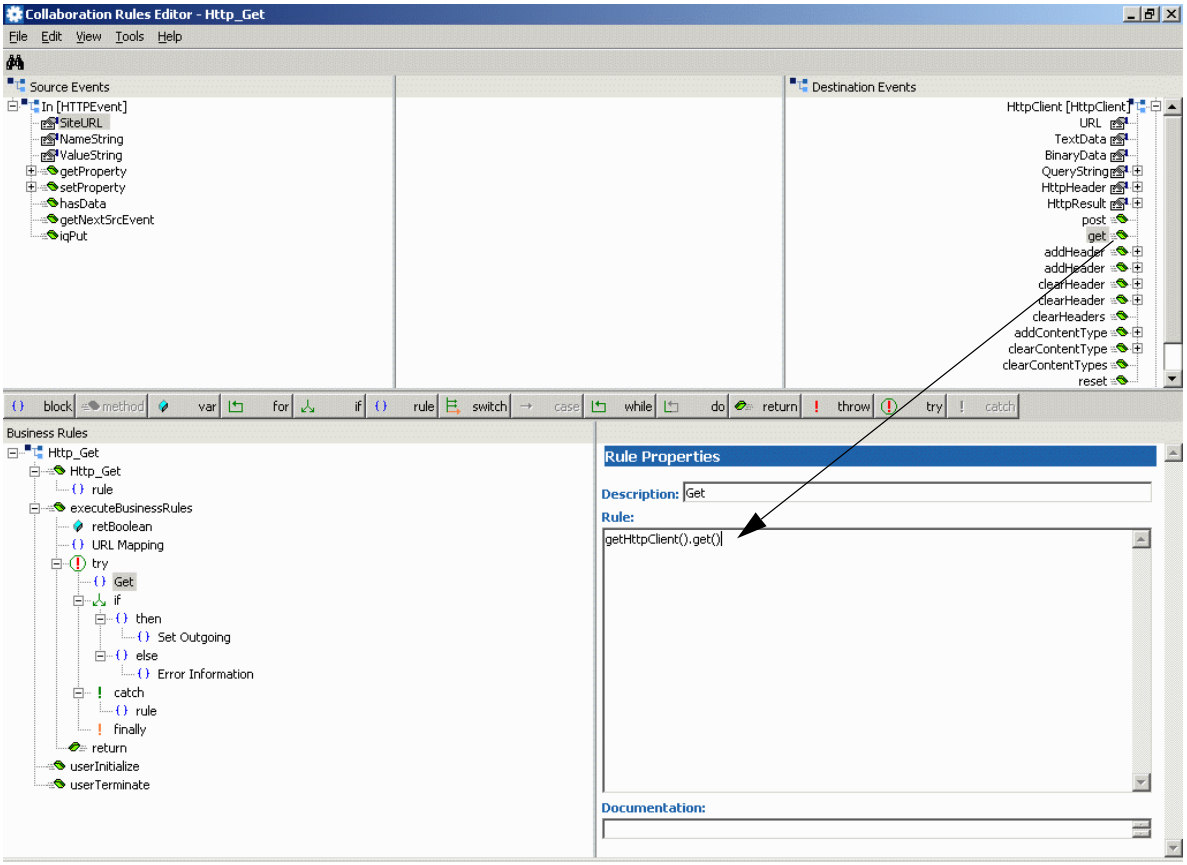
5 Click the **try** conditional expression, it appears below the **UrlMapping** rule.

6 With the **try** conditional expression selected, click **rule**. When asked whether peer or child, select **child** (see Figure 11).

Figure 11 Child or Peer Dialog Box

7 With the newly created rule selected, drag and drop the **get** method from the **HttpClient** destination Event to the Rule Properties pane (see Figure 12).

Figure 12 Collaboration Rules: Mapping Properties Part 1



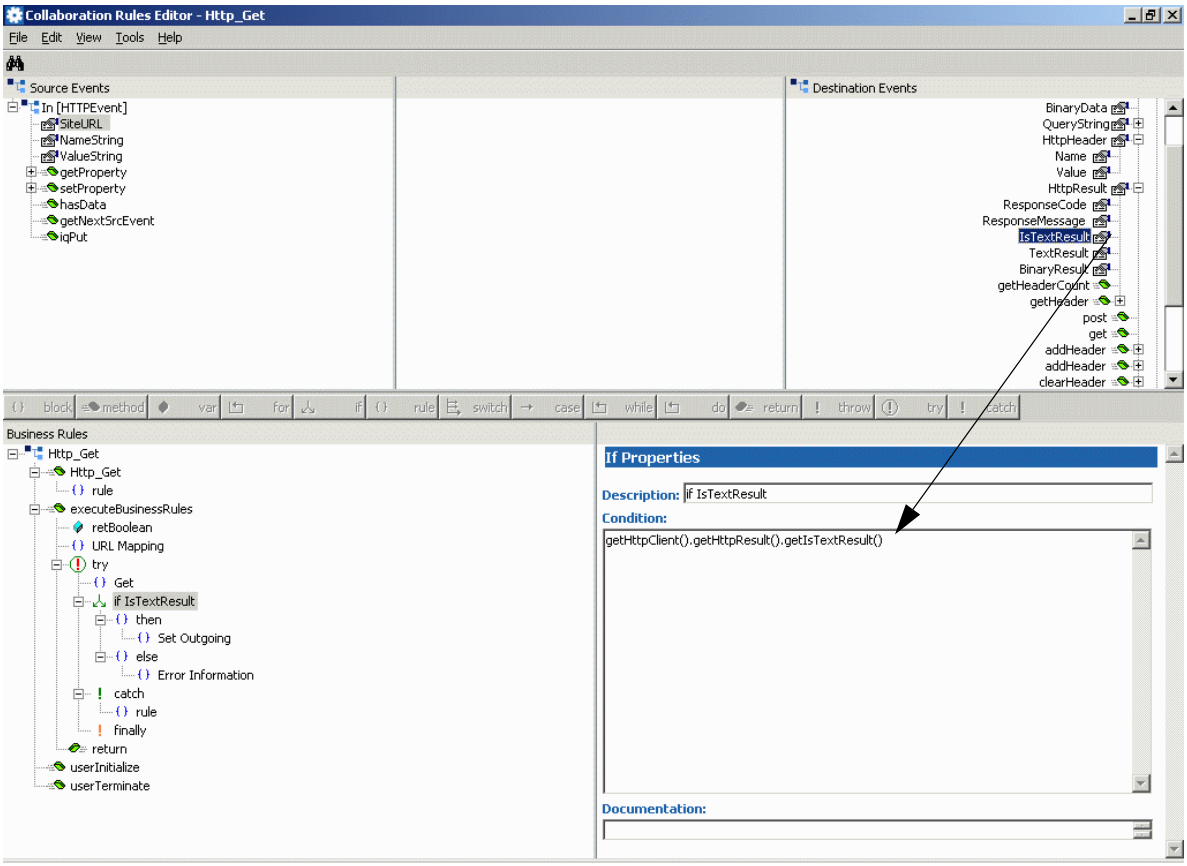
The method appears in the **Rules Properties** dialog box. Change the description from **rule** to **get** (method).

8 With the **get** method selected, click the **if** conditional expression. Drag and drop the **Is TextResult** to the **Condition** dialog box. Ensure that the condition line reads:

```
(getHttpClient().getHttpClient().getIsTextResult())
```

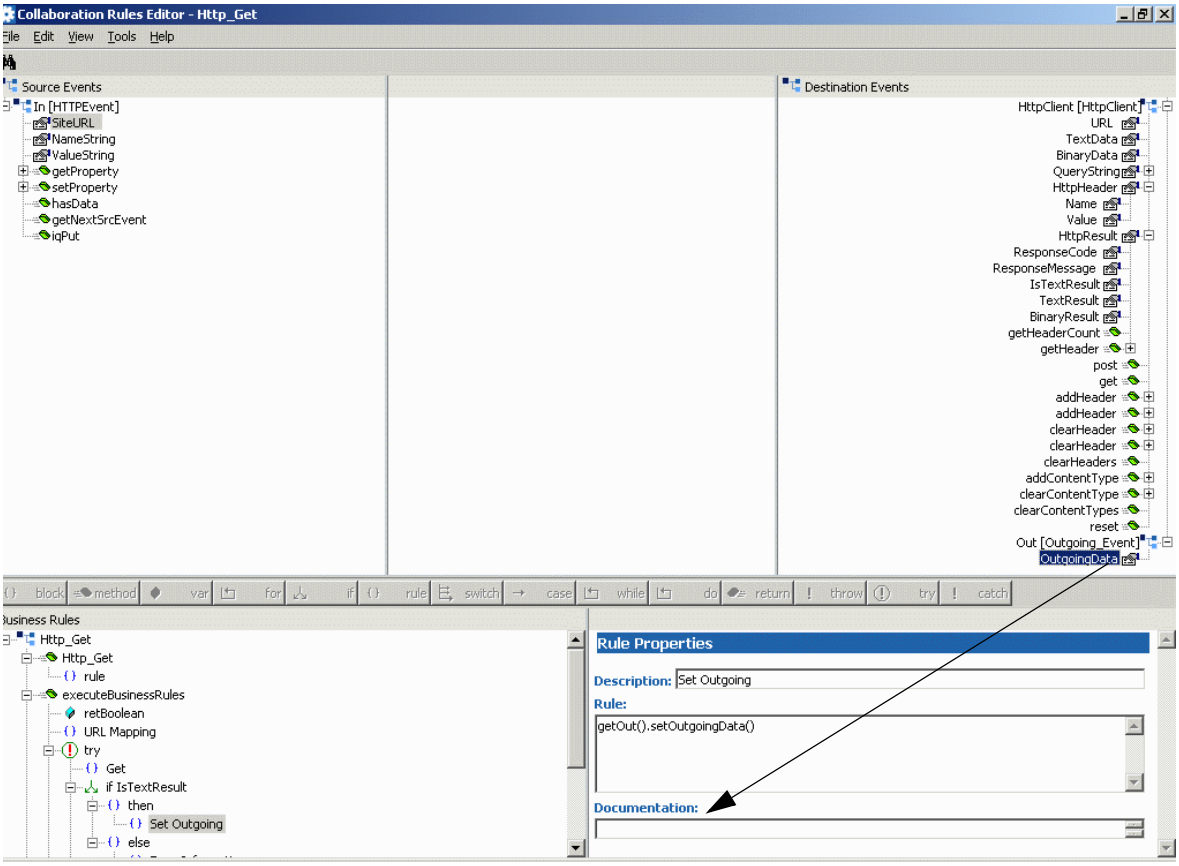
9 Add **Is TextResult** to the description of the conditional expression (see **Figure 13 on page 57**).

Figure 13 Collaboration Rules: Mapping Properties Part 2



10 Select the **then** condition, then click **rule**. Drag and drop the **OutgoingData** property node to the **Documentation** text box (see Figure 14).

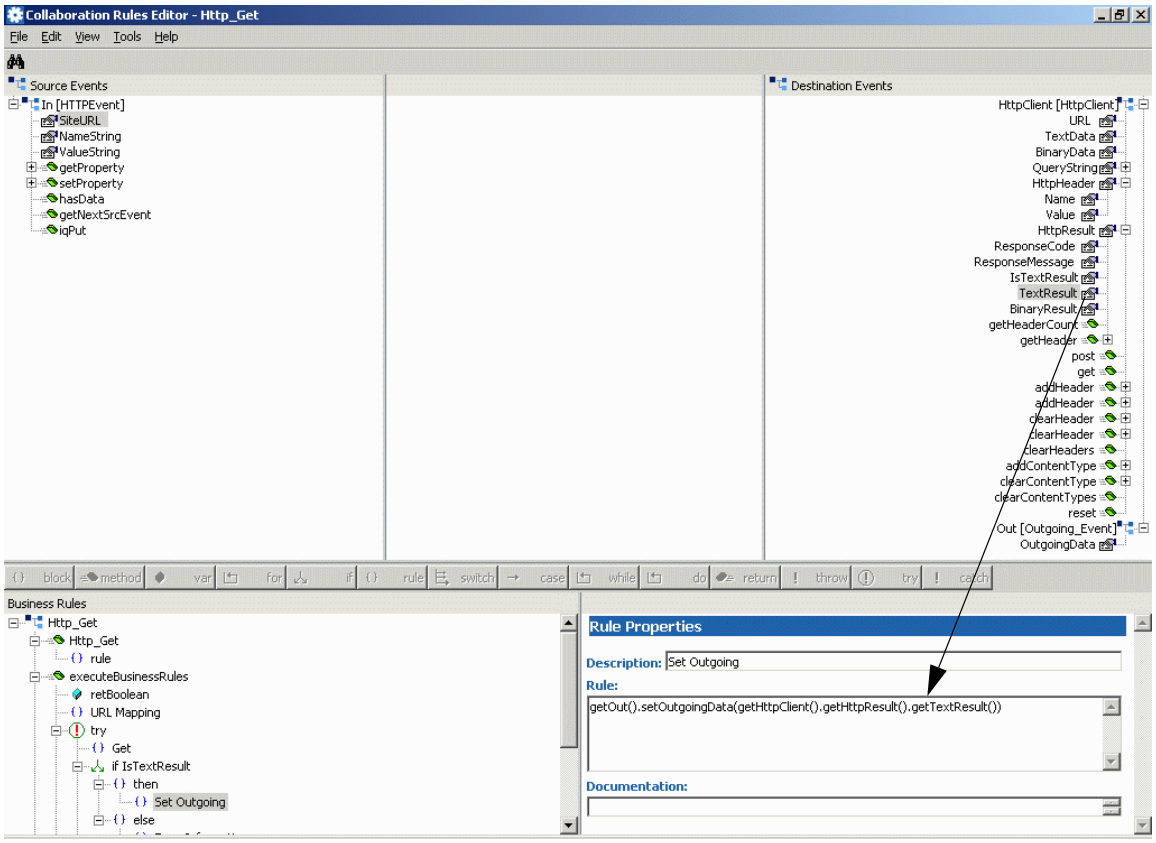
Figure 14 Collaboration Rules: Dragging OutgoingData



11 Change the **Description** text box to read to **SetOutgoingData.**

12 Drag and drop the **TextResult** node into the **setOutgoingData** method as shown in Figure 15. Before releasing the node, ensure that the cursor is placed between the appropriate pair of parentheses for the method.

Figure 15 Business Rules: Part 1



13 Ensure that the condition line reads as follows:

```
getOut().setOutgoingData(getHttpClient().getHttpResult()
    .getTextResult())
```

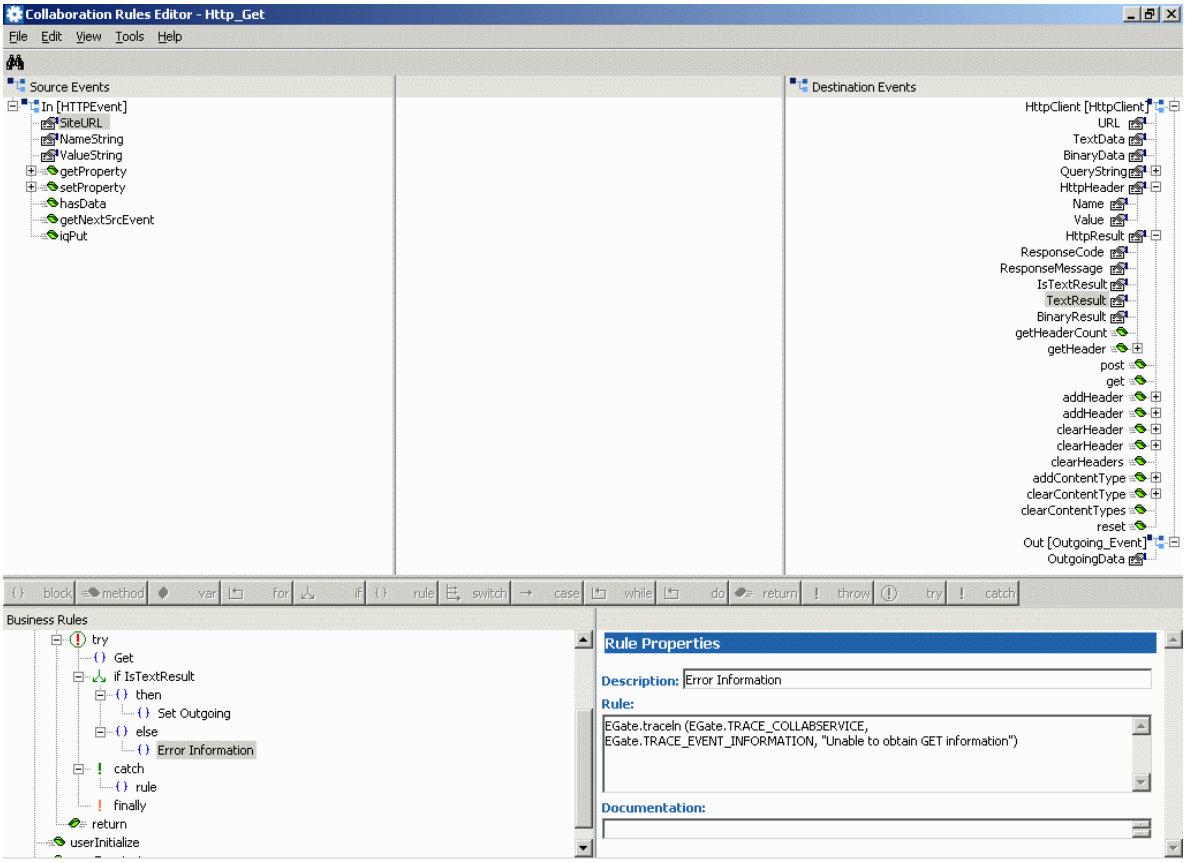
14 Select the **else** condition, then click **rule**.

15 Enter the desired trace information as follows:

```
EGate.traceln (EGate.TRACE_COLLABSERVICE,
EGate.TRACE_EVENT_INFORMATION, "Unable to obtain GET information")
```

See **Figure 16 on page 60**.

Figure 16 Business Rules: Part 2



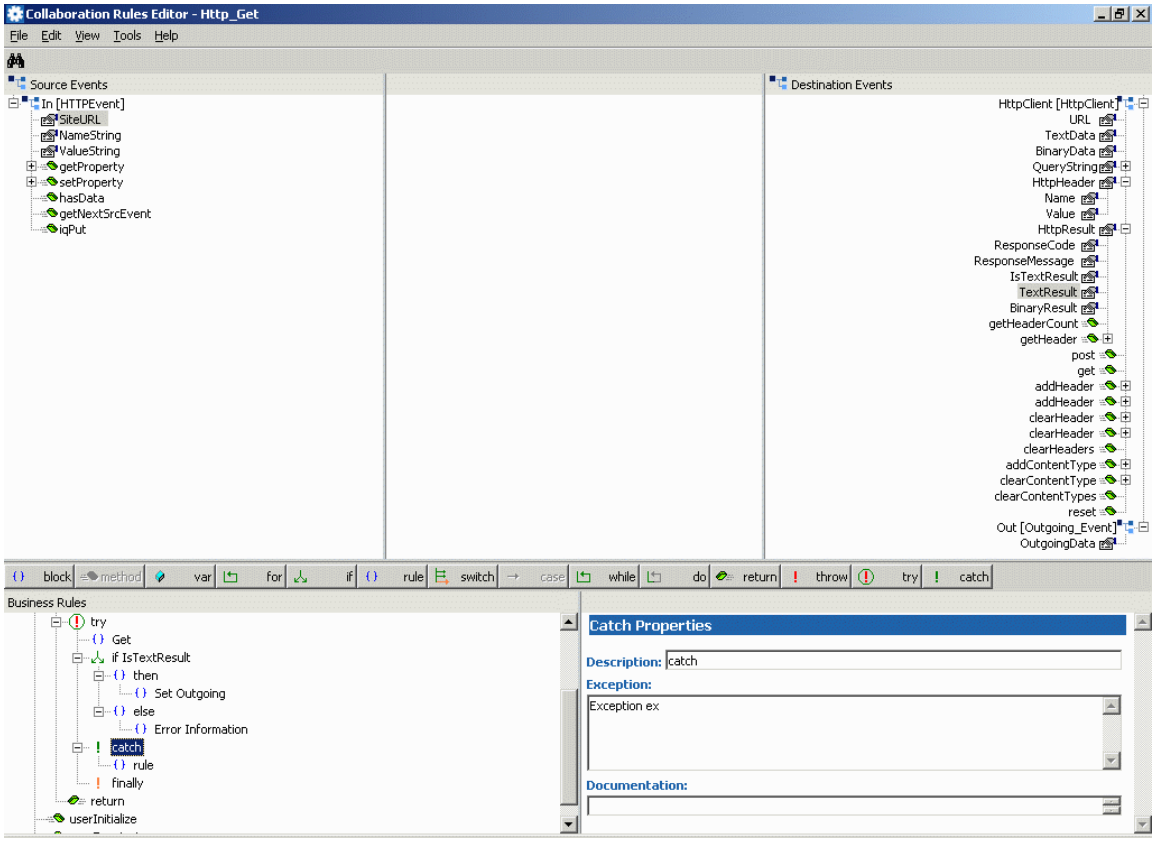
16 Select **try**, then click **catch**.

17 Select the **child** node. In the **Exception** text box, type:

```
Exception ex
```

See **Figure 17 on page 61**.

Figure 17 Business Rules: Part 3



18 Select **catch**, then click **rule**.

19 In the **Rule** text box type:

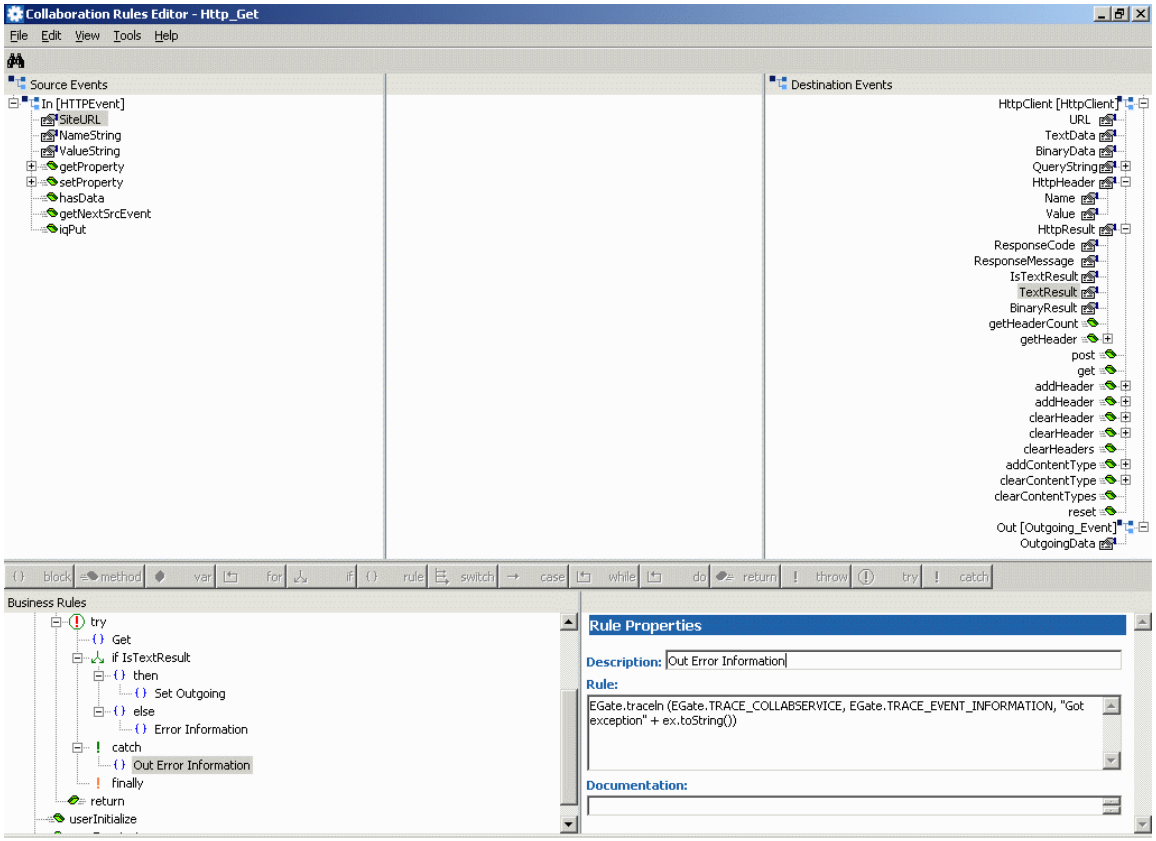
```
EGate.traceln (EGate.TRACE_COLLABSERVICE,
    EGate.TRACE_EVENT_INFORMATION,
    "Got exception" + ex.toString())
```

Be sure to type the text all on one line with a space after each comma.

20 Change the name of the rule to **Output Error Information**.

See **Figure 18 on page 62**.

Figure 18 Business Rules: Part 4

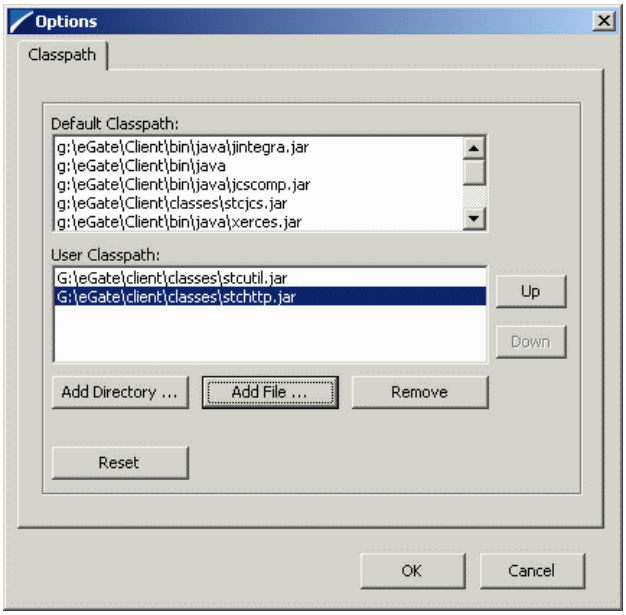


21 Before compiling the code, on the **Tools** menu, click **Options**.

22 Verify that all necessary **.jar** files (see **Figure 19 on page 63**) are included as follows:

- For clear HTTP (SSL not enabled), ensure that the **stchttp.jar** and **stcutils.jar** file are added.
- For a Collaboration that uses SSL, ensure that **jcert.jar**, **jnet.jar**, and **jsse.jar** are included.

Figure 19 Business Rules Options Dialog Box



23 When all the business logic has been defined, the code can be compiled by selecting **Compile** from the **File** menu.

The **Save** menu opens.

24 Provide a name for the **.xpr** file. For this example, use **Http_Get.xpr**.

If the code compiles successfully, the message **Compile Completed** appears. If the outcome is unsuccessful, a Java Compiler error message appears.

25 Once the compilation is complete, save the Collaboration Rules file and exit the Collaboration Rules Editor.

26 Click **OK** on the **Collaboration Rules Properties** dialog box, to save your Collaboration Rules component.

27 In the e*Gate Schema Designer Main window, open the **Collaboration Rules** folder in the Navigation pane. Check the following information:

- The path for the .**class** file created appears in the Editor pane. For this example, the path **collaboration_rules\Http_Class_Get.class** appears.
- Under **Initialization** file, the path for the .**ctl** file created appears. For this example the path **collaboration_rules\Http_Class_Get.ctl** appears.

28 You need to create the **PassIn** and **PassOut** Collaboration Rules. To do so:

- Create and name each Collaboration Rules component (**PassIn** and **PassOut)**.
- Open the **Collaboration Properties** dialog box (**General** tab) and select the **Pass Through** Service for both Collaboration Rules.
- On the **Collaboration Properties** dialog box click **OK** to save both Collaboration Rules.

No further configuration is required.

5.3.9 Creating Collaborations

Collaborations are the components that receive and process Event Types, then forward the output to other e*Gate components or an external system.

Collaborations consist of the subscriber, which receives Events of a known type (sometimes from a given source), and the publisher, which distributes transformed Events to a specified recipient.

Inbound_eWay Collaboration

To create the Inbound_eWay Collaboration

1. In the e*Gate Schema Designer, select the Navigator's **Components** tab.
2. Open the host on which you want to create the Collaboration.
3. Select the Control Broker for this schema.
4. Select the **Inbound_eWay** to assign the Collaboration.
5. On the palette, click the Collaboration icon.
6. Enter the name (**PassIn**) of the new Collaboration, then click **OK**.
7. Select the new Collaboration, then right-click to edit its properties.

   The **Collaboration Properties** dialog box appears.
8. From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously (**PassIn**) for this Collaboration.
9. In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration subscribes. Then do the following operations:
   A. From the **Event Type** list, select the Event Type that you previously defined (**HttpEvent**).
   B. Select the **Source** from the **Source** list. In this case, it should be **<External>**.
10. In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration publishes. Then do the following operations:
    A. From the **Event Types** list, select the Event Type that you previously defined (**HttpEvent**).
    B. Select the publication destination from the **Destination** list. In this case, it should be **<iq_standard>**.

HTTP_Multi_Mode Collaboration

To create the HTTP_Multi_Mode Collaboration

1. In the e*Gate Schema Designer, select the Navigator's **Components** tab.
2. Open the host on which you want to create the Collaboration.
3. Select the Control Broker for this schema.
4. Select the **HTTP_Multi_Mode** to assign the Collaboration.

5 On the palette, click the Collaboration icon.

6 Enter the name (**collab_HTTP**) of the new Collaboration, then click **OK**.

7 Select the new Collaboration, then right-click to edit its properties.

The **Collaboration Properties** dialog box appears.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file (**Http_Get.xpr**) that you created previously for this Collaboration.

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration subscribes. Then do the following operations:

A From the **Instance Name** list, select the instance name that you previously defined (**In**).

B From the **Event Type** list, select the **Event Type** that you previously defined (**HttpEvent**).

C Select the **Source** from the **Source** list. In this case, it should be **<PassIn>**.

10 In the **Publications** area, click **Add** to define the output Event Types that this Collaboration publishes. Then do the following operations:

A From the **Instance Name** list, select the instance name that you previously defined **HttpClient.**

B From the **Event Types** list, select the Event Type that you previously defined (**HttpClient**).

C Select the publication destination from the **Destination** list. In this case, it is **<SimpleHttpCP>**.

11 In the **Publications** area, click **Add** to define the additional output Event Types that this Collaboration publishes.

A From the **Instance Name** list, select the instance name that you previously defined **Out.**

B From the **Event Types** list, select the Event Type that you previously defined (**OutgoingEvent**).

12 Click **OK** to exit.

Outbound_eWay Collaboration

To create the Outbound_eWay Collaboration

1 In the e*Gate Schema Designer, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select the Control Broker for this schema.

4 Select the **Outbound_eWay** to assign the Collaboration.

5 On the palette, click the Collaboration icon.

6 Enter the name of the new Collaboration (**PassOut**), then click **OK**.

7 Select the new Collaboration, then right-click to edit its properties.

The **Collaboration Properties** dialog box appears.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously (**PassOut**) for this Collaboration.

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration subscribes. Then do the following operations:

A From the **Event Type** list, select the Event Type that you previously defined (**HttpClient**).

B Select the **Source** from the **Source** list. In this case, it is **<collab_HTTP>**.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration publishes. Then do the following operations:

A From the **Event Types** list, select the Event Type that you previously defined (**HttpClient**).

B Select the publication destination from the **Destination** list. In this case, it is **<External>**.

After You Are Finished

When you are finished check and test the schema to be sure you have configured it correctly. Now you are ready to run the schema. For details on this operation, see **"Running a Schema" on page 40**.

Chapter 6

Secure Sockets Layer Operation

This chapter explains the operation of the Secure Sockets Layer (SSL) feature available with the HTTP(S) e*Way Intelligent Adapter.

6.1 Using Secure Sockets Layer: Overview

The e*Way's SSL feature offers hyper-text transfer protocol (HTTP) data exchanges security from interception, hackers, and other types of breaches. HTTP with SSL is called HTTP(S), meaning that SSL is enabled and provides security for any HTTP(S) data exchange.

The SSL feature is supported through the use of JSSE version 1.0.2. Currently, the JSSE reference implementation is used. JSSE is a provider-based architecture, meaning 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 interacting 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 implementations cannot support a particular algorithm.

The JSSE 1.0.2 application programming interface (API) is capable of supporting SSL versions 2.0 and 3.0 and Transport Layer Security (TLS) version 1.0. These security protocols encapsulate a normal bidirectional stream socket and the JSSE 1.0.2 API adds transparent support for authentication, encryption, and integrity protection. The JSSE reference implementation implements SSL version 3.0 and TLS 1.0. It does not implement SSL 2.0.

For more information, see the following Web site:

http://java.sun.com/products/jsse/doc/guide/API_users_guide.html

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

6.2 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 certification 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.

6.2.1 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 26** 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 26** for more information.

6.2.2 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. For example, the sample provided on the installation CD-ROM uses **development.seebeyond.com** (see **Chapter 5**). 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, a CA 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, see the following Web sites:

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 25** for more information.

6.2.3 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 26**).

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.

6.3 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 20 on page 73** shows a diagram of the SSL handshake dialog for server-side authentication.
- **Figure 21 on page 74** shows a diagram of the SSL handshake dialog for dual authentication.

Figure 20 Server-side Authentication

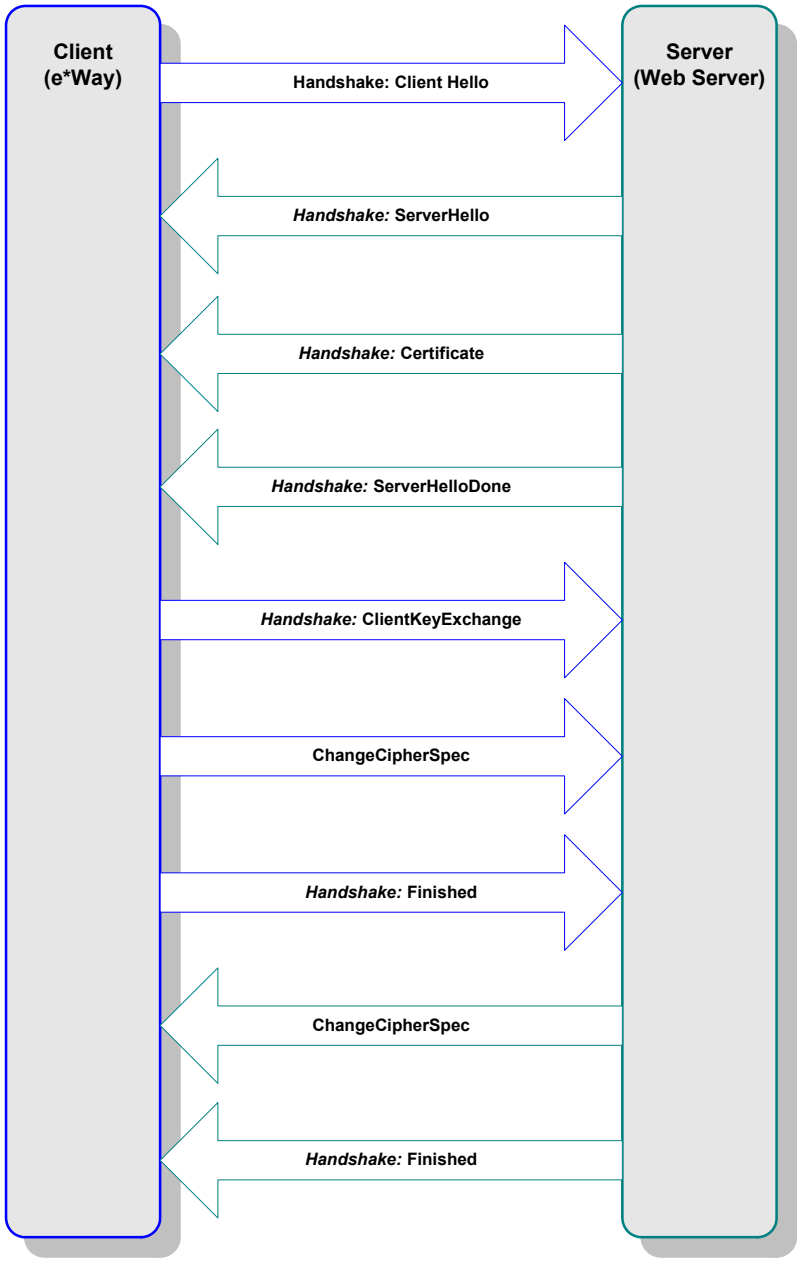


Figure 21 Dual Authentication

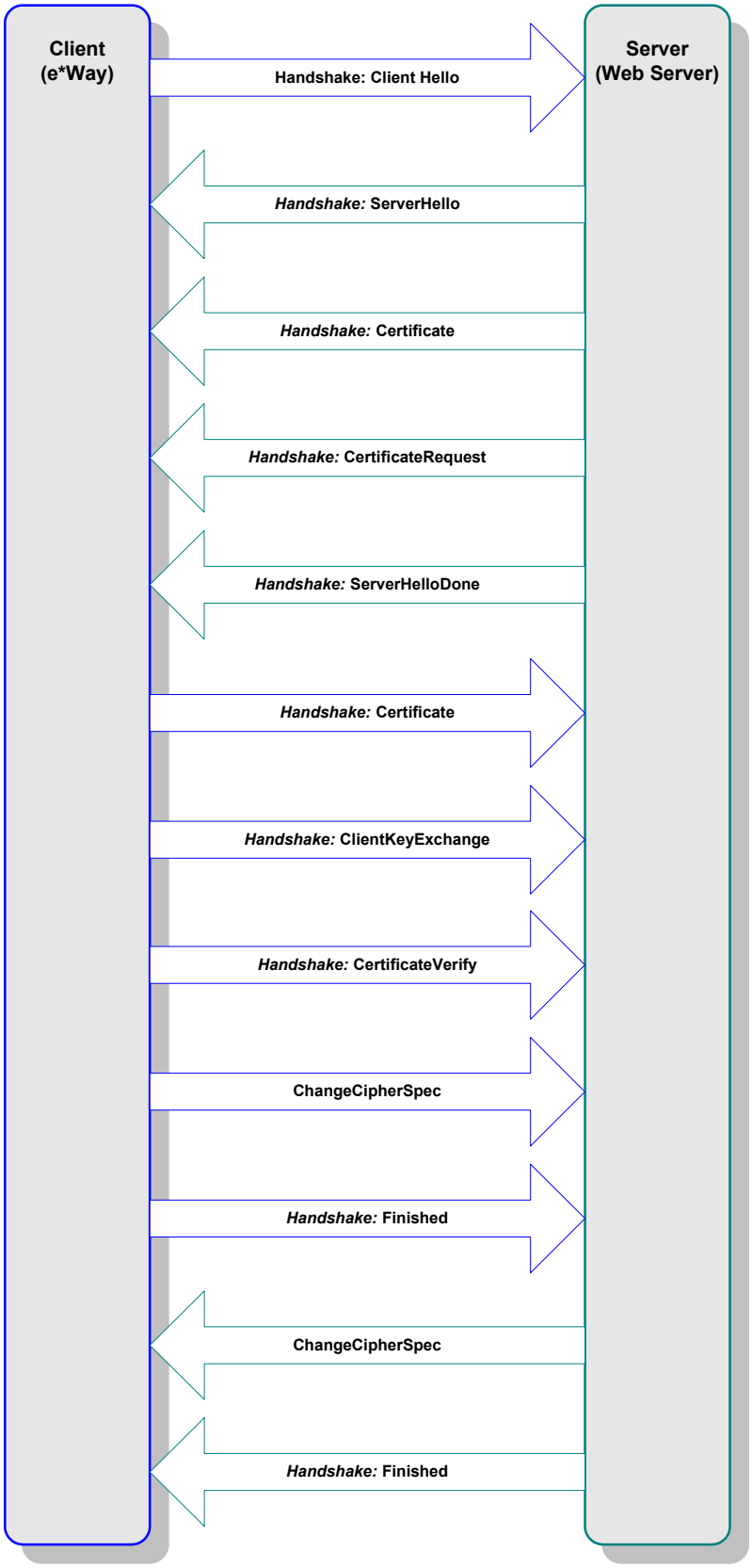
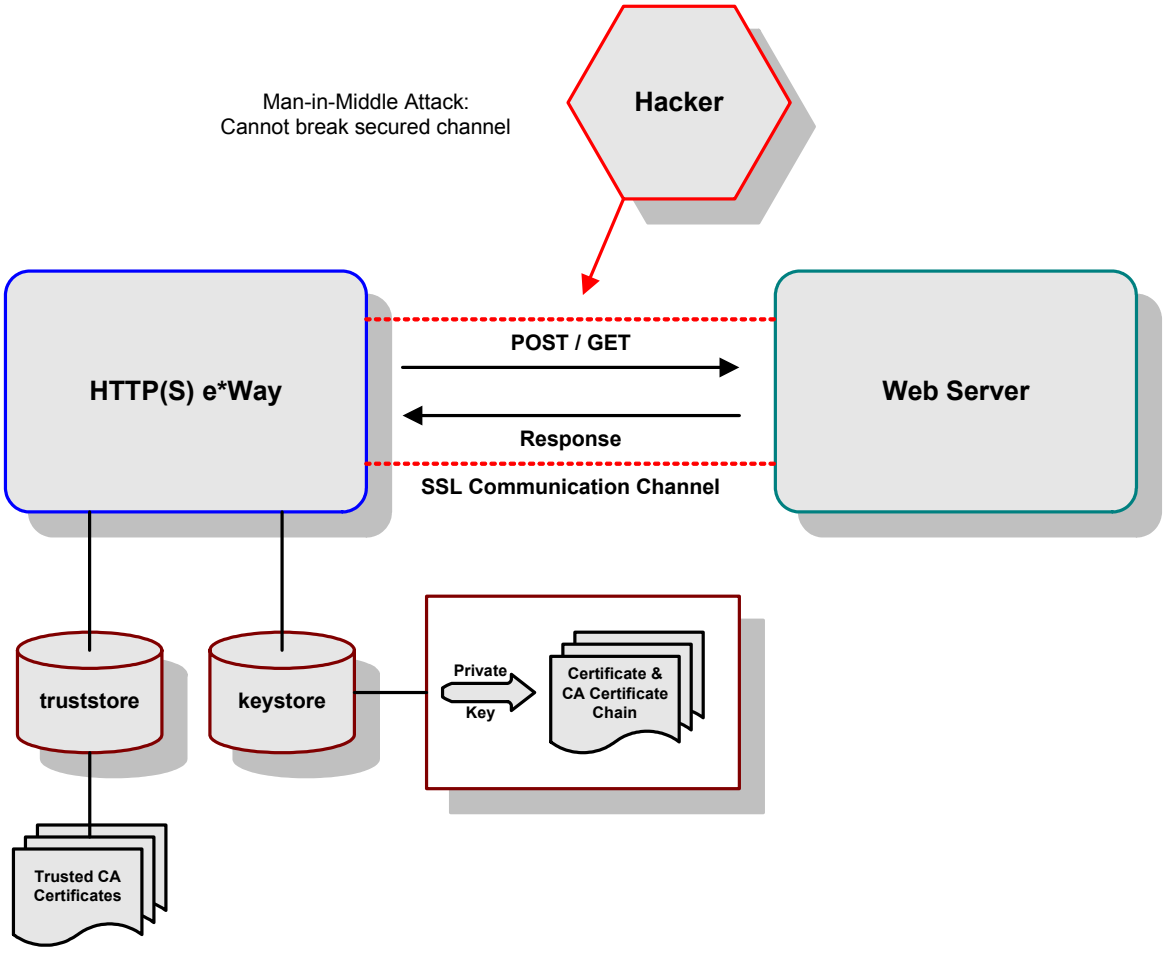


Figure 22 shows a diagram of general SSL operation with the HTTP(S) e*Way.

Figure 22 General SSL Operation: HTTP(S) e*Way



Chapter 7

Java Classes and Methods

This chapter provides an overview of the Java classes and methods contained in the HTTP(S) e*Way Intelligent Adapter, which are used to extend the functionality of the e*Way.

7.1 HTTP(S) e*Way Methods and Classes: Overview

For any e*Way, communication takes place both on the e*Gate Integrator 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 and the external system is different for each e*Way.

For the HTTP(S) e*Way, the **stceway.exe** file (creates a Multi-Mode e*Way; see **Chapter 4**) is used to communicate between the e*Way and e*Gate. A Java Collaboration is utilized to keep the communication open between the e*Way and the external system or network.

7.2 Using Java Methods

Java methods have been added to make it easier to set information in the HTTP(S) e*Way Event Type Definitions (ETDs), as well as get information from them. The nature of this data transfer depends on the configuration parameters (see **Chapter 3**) you set for the e*Way in the e*Gate Schema Designer's e*Way Configuration Editor window.

Note: *For more information on the HTTP(S) e*Way ETD structure and implementation with the e*Way, see* **Chapter 5**.

The Schema Designer's Collaboration Rules Editor window allows you to call Java methods by dragging and dropping an ETD node into the **Rules** scroll box of the **Rules Properties** window.

Note: *The node name can be different from the Java method name.*

After you drag and drop, the actual conversion takes place in the **.xsc** file. To view the **.xsc** file, use the Schema Designer's ETD Editor or Collaboration Rules Editor windows.

For example, if the node name is **HttpPassWord**, the associated **javaName** is **HttpPassWord**. If you want to get the node value, use the Java method called **getHttpPassWord()**. If you want to set the node value, use the Java method called **setHttpPassWord()**.

The HTTP(S) e*Way has the following Java classes:

7.3 HttpAuthenicator Class

The **HttpAuthenicator** class constructs an **HttpAuthenticator**, and extends **java.net.Authenticator**.

The **HttpAuthenticator** class is defined as:

```
public class HttpAuthenticator
```

The **HttpAuthenticator** class extends **java.net.Authenticator**

The **HttpAuthenticator** class methods include:

register

Description

register registers this HTTP authentication instance for Proxy and/or HTTP authentication.

Syntax

```
public void register(httpPassWord)
```

Parameters

Name	Type	Description
httpPassWord	java.lang.String	The password for HTTP authentication.

Returns

None.

Throws

java.lang.SecurityException

setHttpPassWord

Description

setHttpPassWord sets the password for HTTP authentication.

Syntax

```
public void setHttpPassWord(java.lang.String httpPassWord)
```

Parameters

Name	Type	Description
httpPassWord	java.lang.String	The password for HTTP authentication.

Returns

None.

Throws

None.

setHttpUserName

Description

setHttpUserName sets the user name for HTTP authentication.

Syntax

```
public void setHttpUserName(java.lang.String httpUserName)
```

Parameters

Name	Type	Description
httpUserName	java.lang.String	The user name for HTTP authentication.

Returns

None.

Throws

None.

setProxyHost

Description

setProxyHost sets the proxy host so that when proxy authentication is requested, this authenticator can send the appropriate user name and password to the proxy host.

Syntax

```
public void setProxyHost(java.lang.String proxyHost)
```

Parameters

Name	Type	Description
proxyHost	java.lang.String	The host name or IP address (dotted quad address).

Returns

None.

Throws

java.net.UnknownHostException: When the IP address or host provided cannot be determined.

setProxyPassWord

Description

setProxyPassWord sets the Proxy password for Proxy authentication..

Syntax

```
public void setProxyPassWord(java.lang.String proxyPassWord)
```

Parameters

Name	Type	Description
proxyPassWord	java.lang.String	The user name for Proxy authentication.

Returns

None.

Throws

None.

---

setProxyUserName

Description

setProxyUserName sets the Proxy user name for Proxy authentication.

Syntax

```
public void setProxyUserName(java.lang.String proxyUserName)
```

Parameters

Name	Type	Description
proxyUserName	java.lang.String	The user name for Proxy authentication.

Returns

None.

Throws

None.

7.4 HttpClient Class

The **HttpAuthenicator** class constructs an HTTP Client.

The **HttpClient** class is defined as:

```
public class HttpClient
```

The **HttpClient** class methods include:

addContentType

Description

addContentType adds a Content-Type value, so that the next request sent contains the specified value in the Content-Type header.

Syntax

```
public void addContentType(java.lang.String contentType)
```

Parameters

Name	Type	Description
contentType	java.lang.String	The string specifying the type of message content of the request.

Returns

None.

Throws

None.

addHeader

Description

addHeader adds the specified header, so that the next request sent contains the specified header information.

Syntax

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

Parameters

Name	Type	Description
header	object	The header to be added.
name	java.lang.String	The NAME portion of the of the header to be added.
value	java.lang.String	The VALUE portion of the of the header to be added.

Returns

None.

Throws

None.

clearContentType

Description

clearContentType removes the Content-Type value if the specified Content-Type was previously added, so that the next request sent does not contain the specified Content-Type value.

Syntax

```
public void ClearContentType(java.lang.String contentType)
```

Parameters

Name	Type	Description
contentType	java.lang.String	The Content Type to remove.

Returns

None.

Throws

None.

clearContentTypes

Description

clearContentTypes removes all previously added Content-Type values, so that the next request sent does not contain any of the previously added Content-Type information.

Syntax

```
public void clearContentsTypes()
```

Parameters

None.

Returns

None.

Throws

None.

clearHeader

Description

clearHeader removes the specified header if the header was previously set, so that the next request sent does not contain the specified header information.

Syntax

```
public void clearHeader(HttpHeader header, java.lang.String.name)
```

Parameters

Name	Type	Description
header	object	The header to remove from the list of headers.
name	java.lang.String	The name of the header used to locate the HTTP header for removal from the list of headers.

Returns

None.

Throws

None.

clearHeaders

Description

clearHeaders removes all previously added headers, so that the next request sent does not contain any of the previously added header information.

Syntax

```
public void clearHeaders()
```

Parameters

None.

Returns

None.

Throws

None.

---

get

Description

get performs an HTTP or HTTP(S) GET method. The **get** method can also send form data if you have set the form data via the **setQueryString** method.

Syntax

```
public void get()
```

Parameters

None.

Returns

None.

Throws

java.lang.Exception

java.io.IOException: When an exception is returned from attempting to open an input stream from the connection.

java.net.MalformedURLException: When the protocol in the URL specified is not a legal protocol or if the URL string could not be parsed.

Additional Information

For more information, see **setQueryString** on page 94 and **setURL** on page 95.

---

getBinaryData

getBinaryData gets the binary data to be posted that was previously set by **setBinaryData**.

Syntax

```
public byte[] getBinaryData()
```

Parameters

None.

Returns

Byte Array

The text data to be posted; returns null if binary data was not previously set.

Throws

None.

getHttpAuthenticator

Description

getHttpAuthenticator gets the current HTTP authenticator object that was previously set with **setHttpAuthenticator** method.

Syntax

public HttpAuthenticator **getHTTPAuthenticator**()

Parameters

None.

Returns

Object

The HTTP authenticator object that was previously set or null, if not previously set.

Throws

None.

getHttpHeader

Description

getHttpHeader gets the HttpHeader placeholder object. Added to work with the .**xsc** file and the java collaboration editor for drag and drop. Once data is populated in the HttpHeader placeholder object, use one of the other methods for managing header information.

Syntax

```
public HttpHeader getHttpHeader()
```

Parameters

None.

Returns

Object

The HttpHeader object being used as a placeholder for adding HTTP header information.

Throws

None.

getHttpProxyHost

Description

getHttpProxyHost gets the current HTTP proxy host that was previously set.

Syntax

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

Parameters

None.

Returns

java.lang.String

The name of the HTTP proxy host currently being used or null if HTTP proxy host was not set.

Throws

None.

getHttpProxyPort

Description

getHttpProxyPort gets the current HTTP proxy port that was previously set.

Syntax

```
public int getHttpProxyPort()
```

Parameters

None.

Returns

Integer

The HTTP proxy port currently being used or **-1** if HTTP proxy host was not set.

Throws

None.

getHttpResult

Description

getHttpResult gets the **HttpResult** placeholder object.

Syntax

```
public HttpResult getHttpResult()
```

Parameters

None.

Returns

Object

The **HttpResult** placeholder object used for setting and getting header information.

Throws

None.

---

getHttpsProxyHost

Description

getHttpsProxyHost gets the current HTTP(S) proxy host that was previously set.

Syntax

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

Parameters

None.

Returns

java.lang.String

The name of the HTTP(S) proxy host currently being used or null if HTTP(S) proxy host was not set.

Throws

None.

---

getHttpsProxyPort

Description

getHttpsProxyPort gets the current HTTP(S) proxy port that was previously set.

Syntax

```
public int getHttpsProxyPort()
```

Parameters

None.

Returns

Integer

The HTTP(S) proxy port currently being used or **-1** if HTTP(S) proxy port was not set

Throws

None.

getQueryString

Description

getQueryString gets the query data that was previously set by **setQueryString**.

Syntax

```
public String getQueryString()
```

Parameters

None.

Returns

Object

The text data to be posted; returns null if form data was not previously set.

Throws

None.

getTextData

Description

getTextData gets the text data to be posted that was previously set by **setTextData**.

Syntax

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

Parameters

None.

Returns

java.lang.String

The text data to be posted; returns null if text data was not previously set.

Throws

None.

getURL

Description

getURL gets the URL.

Syntax

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

Parameters

None.

Returns

java.lang.String

The URL string previously set by setURL.

Throws

None.

initialize

Description

initialize initializes an object when called by an external Collaboration Service.

Syntax

```
public void initialize(com.stc.eways.http.JCollabController
    cntrCollab, java.lang.String key, int mode)
```

Parameters

Name	Type	Description
cntrCollab	object	The Java Collaboration Controller object.
key	java.lang.String	The initialization key.
mode	integer	The mode of initialization.

Returns

None.

Throws

com.stc.eways.http.CollabConnException

com.stc.eways.http.CollabDataException

post

Description

post posts the data previously set by one of the following methods: **setTextData, setQueryString,** and **setBinaryData**. The Content-Type can be changed with **setContentType**() before posting the data.

Syntax

```
public void post()
```

Parameters

None.

Returns

None.

Throws

java.lang.String.Exception

java.io.IOException: Thrown when an exception is returned from attempting to open an input stream from the connection.

java.net.MalformedURLException: Thrown when the protocol in the URL specified is not a legal protocol or the URL string could not be parsed.

reset

Description

reset clears all headers and request data from the memory.

Syntax

```
public boolean reset()
```

Parameters

None.

Returns

Boolean

Returns **true** if the call is successful, **false** if not.

Throws

None.

setBinaryData

Description

setBinaryData sets the raw binary data to be posted.

Syntax

```
public void setBinaryData(byte[] binaryData)
```

Parameters

Name	Type	Description
binaryData	byte array	The binary data to be posted.

Returns

None.

Throws

None.

setCookie

Description

setCookie enables or disables cookies.

Syntax

```
public void setCookie(boolean allowCookies)
```

Parameters

Name	Type	Description
allowCookies	boolean	Set to TRUE to allow cookies; otherwise, set to false to disable cookies.

Returns

None.

Throws

None.

setHttpAuthenticator

Description

setHttpAuthenticator sets the HTTP Authenticator object for use with Web sites that require user name and password authentication.

Syntax

```
public void setHttpAuthenticator(HttpAuthenticator httpAuthenticator)
```

Parameters

Name	Type	Description
httpAuthenticator	object	The HTTP Authenticator object.

Returns

None.

Throws

None.

setHttpHeader

setHttpHeader sets the HttpHeader placeholder object. Added to work with .**xsc** files and the Java Collaboration Editor for drag and drop. Once data is populated in the HttpHeader placeholder object, use one of the other methods for managing header information. If this method is not called, a default HttpHeader object is used.

Syntax

```
public void setHttpHeader(HttpHeader header)
```

Parameters

Name	Type	Description
header	object	The HttpHeader object to be used as a placeholder for adding HTTP header information.

Returns

None.

Throws

None.

setHttpProxyHost

Description

setHttpProxyHost sets the HTTP proxy host.

Syntax

```
public void setHttpProxyHost(java.lang.String httpProxyHost)
```

Parameters

Name	Type	Description
httpProxyHost	java.lang.String	The HTTP proxy host to use.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP proxy host.

setHttpProxyPort

Description

setHttpProxyPort sets the HTTP proxy port.

Syntax

```
public void setHttpProxyPort(int httpProxyPort)
```

Parameters

Name	Type	Description
httpProxyPort	integer	The HTTP proxy port to use on the HTTP proxy host set with setHttpProxyHost.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP Proxy Port.

setHttpResult

Description

setHttpResult sets the HttpResult placeholder object. Added to work with **.xsc** and Java Collaboration Editor for drag and drop. Use the HttpResult placeholder object to retrieve data from the HTTP server. If this method is not called, a default HttpResult object is used.

Syntax

```
public void setHttpResult(HttpResult result)
```

Parameters

Name	Type	Description
result	object	The HttpResult object to be used as a placeholder for retrieving HTTP results from the HTTP server.

Returns

None.

Throws

None.

setHttpsProxyHost

Description

setHttpsProxyHost sets the HTTP(S) proxy host.

Syntax

```
public void setHttpsProxyHost(java.lang.String httpsProxyHost)
```

Parameters

Name	Type	Description
httpsProxyHost	java.lang.String	The HTTP(S) proxy host to use.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP(S) Proxy Host.

setHttpsProxyPort

Description

setHttpsProxyPort sets the HTTP(S) proxy port.

Syntax

```
public void setHttpsProxyPort(int httpsProxyPort)
```

Parameters

Name	Type	Description
httpsProxyPort	integer	The HTTP(S) proxy port to use on the HTTP(S) proxy host set with setHttpsProxyHost.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP(S) Proxy Port.

setQueryString

Description

setQueryString sets the ETD's **QueryString** node for a query.

Syntax

```
public void setQueryString(QueryString query)
```

Parameters

Name	Type	Description
query	QueryString	The QueryString that contains the URL-encoded name value pairs.

Returns

None.

Throws

None.

Additional Information

For more information, see get on page 84.

setTextData

Description

setTextData sets the raw text data to be posted.

Syntax

```
public void setTextData(java.lang.String textData)
```

Parameters

Name	Type	Description
textData	java.lang.String	The text data to be posted.

Returns

None.

Throws

None.

setURL

Description

setURL sets the URL.

Syntax

```
public void setURL(java.lang.String urlString)
```

Parameters

Name	Type	Description
urlString	java.lang.String	The URL string associated with the URL setting. The name value pairs that are supplied in the URL must be URL-encoded. The QueryString class is preferred when doing a Post or Get with multiple name value pairs, rather than hard-coded in the URL.

Returns

None.

Throws

java.net.MalformedURLException: Thrown when an exception is returned in an attempt to construct a java.net.URL object with the supplied URL string.

7.5 HttpClientAPI Class

The **HttpClientAPI** class extends the **java.lang.Object**.

The **HttpClientAPI** class is defined as:

```
public class HttpClientAPI
```

The **HttpClientAPI** class methods include:

addContentType

Description

addContentType adds a Content-Type value, so that the next request sent contains the specified value in the Content-Type header.

Syntax

```
public void addContentType(java.lang.String contentType)
```

Parameters

Name	Type	Description
contentType	java.lang.String	The String specifying the type of message content of the request.

Returns

None.

Throws

None.

addHeader

Description

addHeader adds the specified header, so that the next request sent contains the specified header information.

Syntax

```
public void addHeader(HttpHeader header)
```

Parameters

Name	Type	Description
header	HttpHeader	The header to be added.

Returns

None.

Throws

None.

addHeader

Description

addHeader adds the specified header, so that the next request sent contains the specified header information.

Syntax

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

Parameters

Name	Type	Description
name	java.lang.String	The NAME portion of the header to be added.
value	java.lang.String	The VALUE portion of the header to be added.

Returns

None.

Throws

None.

clearContentType

clearContentType removes, if the specified Content-Type was previously added, the Content-Type value, so that the next request sent does not contain the specified Content-Type value.

Syntax

```
public void clearContentType(java.lang.String contentType)
```

Parameters

Name	Type	Description
contentType	java.lang.String	The Content Type to remove

Returns

None.

Throws

None.

clearContentTypes

Description

clearContentTypes removes all previously-added Content-Type values, so that the next request sent does not contain any previously added Content-Type information.

Syntax

public void **clearContentTypes**()

Parameters

None.

Returns

None.

Throws

None.

clearHeader

Description

clearHeader removes, if the specified header was previously set, the header, so that the next request sent does not contain the specified header information.

Syntax

```
public void clearHeader(HttpHeader header)
```

Parameters

Name	Type	Description
header	HttpHeader	The header to remove from the list of headers.

Returns

None.

Throws

None.

clearHeader

Description

clearHeader removes, if the specified header was previously set, the header, so that the next request sent does not contain the specified header information.

Syntax

```
public void clearHeader(java.lang.String name)
```

Parameters

Name	Type	Description
name	java.lang.String	The name of the header used to locate the HTTP header for removal from the list of headers.

Returns

None.

Throws

None.

clearHeaders

Description

clearHeaders removes all previously added headers, so that the next request sent does not contain any of the previously added header information.

Syntax

```
public void clearHeaders()
```

Parameters

None.

Returns

None.

Throws

None.

get

Description

get retrieves the data previously set by either method **setURL** or **setQueryString**.

Syntax

```
public HttpResult get()
```

Parameters

None.

Returns

Object

An **HttpResponse** structure containing the response

Throws

java.lang.Exception

java.io.IOException: When an exception is returned from attempting to open an input stream from the connection.

java.net.MalformedURLException: When the protocol in the URL specified is not a legal protocol or the URL string could not be parsed.

---

getBinaryData

getBinaryData gets the binary data to be posted that was previously set by setBinaryData.

Syntax

```
public byte[] getBinaryData()
```

Parameters

None.

Returns

Byte Array

The text data to be posted; returns null if binary data was not previously set.

Throws

None.

---

getHttpAuthenticator

Description

getHttpAuthenticator gets the current HTTP authenticator object that was previously set with the **setHttpAuthenticator** method.

Syntax

```
public HttpAuthenticator getHttpAuthenticator()
```

Parameters

None.

Returns

Object

The HTTP authenticator object that was previously set, or null if not previously set.

Throws

None.

getHttpProxyHost

Description

getHttpProxyHost gets the current HTTP proxy host that was previously set.

Syntax

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

Parameters

None.

Returns

java.lang.String

The name of the HTTP proxy host currently being used, or null if HTTP proxy host was not set.

Throws

None.

getHttpProxyPort

Description

getHttpProxyPort gets the current HTTP proxy port that was previously set.

Syntax

```
public int getHttpProxyPort()
```

Parameters

None.

Returns

Integer

The HTTP proxy port currently being used, or **-1** if HTTP proxy host was not set.

Throws

None.

getHttpsProxyHost

Description

getHttpsProxyHost gets the current HTTP(S) proxy host that was previously set.

Syntax

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

Parameters

None.

Returns

java.lang.String

The name of the HTTP(S) proxy host currently being used, or null if HTTP(S) proxy host was not set.

Throws

None.

---

getHttpsProxyPort

Description

getHttpsProxyPort gets the current HTTP(S) proxy port that was previously set.

Syntax

```
public int getHttpsProxyPort()
```

Parameters

None.

Returns

Integer

The HTTP(S) proxy port currently being used, or **-1** if the HTTP(S) host was not set.

Throws

None.

---

getQueryString

getQueryString gets the query data that was previously set by **setQueryString**.

Syntax

```
public QueryString getQueryString()
```

Parameters

None.

Returns

Object

The text data to be posted; returns null if form data was not previously set.

Throws

None.

getTextData

Description

getTextData gets the text data to be posted that was previously set by **setTextData**.

Syntax

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

Parameters

None.

Returns

java.lang.String

The text data to be posted; returns null if text data was not set.

Throws

None.

getURL

Description

getURL gets the URL.

Syntax

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

Parameters

None.

Returns

java.lang.String

The URL string previously set by setURL.

Throws

None.

post

Description

post posts the data previously set by one of the following methods: **setTextData, setQueryString,** and **setBinaryData**. The Content-Type can be changed with **setContentType()** before posting the data.

Syntax

```
public HttpResult post()
```

Parameters

None.

Returns

Object

An **HTTPResponse** structure containing the response.

Throws

java.lang.Exception

java.io.IOException: When an exception is returned from attempting to open an input stream from the connection.

java.net.MalformedURLException: When the protocol in the URL specified is not a legal protocol, or the URL string could not be parsed.

reset

Description

reset clears all headers and request data from memory.

Syntax

```
public void reset()
```

Parameters

None.

Returns

None.

Throws

None.

setBinaryData

Description

setBinaryData sets the raw binary data to be posted.

Syntax

```
public void setBinaryData(byte[] binaryData)
```

Parameters

Name	Type	Description
binaryData	byte array	The binary data to be posted.

Returns

None.

Throws

None.

setCookie

setCookie enables or disables cookies.

Syntax

```
public void setCookie(boolean allowCookie)
```

Parameters

Name	Type	Description
allowCookie	boolean	Set to true to allow cookies; otherwise, set to false to disable cookies.

Returns

None.

Throws

None.

setHttpAuthenticator

Description

setHttpAuthenticator sets the HTTP Authenticator object for use with Web sites that require user name and password authentication.

Syntax

```
public void setHttpAuthenticator(HttpAuthenticator httpAuthenticator)
```

Parameters

Name	Type	Description
httpAuthenticator	object	The HTTP Authenticator object.

Returns

None.

Throws

None.

setHttpProxyHost

Description

setHttpProxyHost sets the HTTP proxy host.

Syntax

```
public void setHttpProxyHost(java.lang.String httpProxyHost)
```

Parameters

Name	Type	Description
httpProxyHost	java.lang.String	The HTTP proxy host to use.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP proxy host.

setHttpProxyPort

Description

setHttpProxyPort sets the HTTP proxy port.

Syntax

```
public void setHttpProxyPort(int httpProxyPort)
```

Parameters

Name	Type	Description
httpProxyPort	integer	The proxy port to use on the HTTP proxy host set with setHttpProxyHost.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP Proxy Port.

setHttpsProxyHost

Description

setHttpsProxyHost sets the HTTP(S) proxy host.

Syntax

```
public void setHttpsProxyHost(java.lang.String httpsProxyHost)
```

Parameters

Name	Type	Description
httpsProxyHost	java.lang.String	The HTTP(S) proxy host to use.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP(S) Proxy Host.

setHttpsProxyPort

Description

setHttpsProxyPort sets the HTTP(S) proxy port.

Syntax

```
public void setHttpsProxyPort(int httpsProxyPort)
```

Parameters

Name	Type	Description
httpsProxyPort	integer	The HTTP(S) proxy port to use on the HTTP(S) proxy host set with setHttpsProxyHost.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set HTTP(S) proxy port.

setQueryString

setQueryString sets the QueryString for a query.

Syntax

```
public void setQueryString(QueryString query)
```

Parameters

Name	Type	Description
query	QueryString	The QueryString that contains the URL-encoded name value pairs.

Returns

None.

Throws

None.

setTextData

Description

setTextData sets the raw text data to be posted.

Syntax

public void **setTextData**(java.lang.String textData)

Parameters

Name	Type	Description
textData	java.lang.String	The text data to be posted.

Returns

None.

Throws

None.

setURL

Description

setURL set the URL.

Syntax

```
public void setURL(java.lang.String urlString)
```

Parameters

Name	Type	Description
urlString	java.lang.String	The name value pairs that are supplied in the URL and must be URL-encoded. The QueryString class is preferred when doing a Post or Get with multiple name value pairs, rather than hard-coded in the URL.

Returns

None.

Throws

java.net.MalformedURLException: Thrown when an exception is returned in an attempt to construct a java.net.URL object with the supplied URL string.

7.6 HttpClientConnector Class

The **HttpClientConnector** class makes a connection to the external HTTP server. The main use of this class is to collect configuration parameters. Unlike a connection to a database, there is no persistent connection to a HTTP server, and, thus, most of the methods do nothing.

The **HttpClientConnector** class is defined as:

```
public class HttpClientConnector
```

The **HttpClientConnector** class methods include:

close

Description

close closes the connector to the external system and releases resources.

Syntax

```
public void close()
```

Parameters

None.

Returns

None.

Throws

com.stc.jcsre.EBobConnectorException: Thrown when connection problems occur.

getProperties

Description

getProperties retrieves the connector properties (stored by the constructor) used by the connector to access the external system.

Syntax

```
public java.util.Properties getProperties()
```

Parameters

None.

Returns

Object

Connection properties of the external system.

Throws

None.

isOpen

Description

isOpen verifies that the connector to the external system is still available.

Syntax

```
public boolean isOpen()
```

Parameters

None.

Returns

Boolean

true if the connector is still open and available; **false** otherwise.

Throws

com.stc.jcsre.EBobConnectionException: Thrown when connection problems occur.

open

Description

open opens the connector for accessing the external system.

Syntax

```
public void open(boolean intoEgate)
```

Parameters

Name	Type	Description
intoEgate	boolean	TRUE if the connector is to subscribe for events initially from an external system and inbound to e*Gate; FALSE if the connector is to publish events outbound from e*Gate to an external system.

Returns

None.

Throws

com.stc.jcsre.EBobConnectionException: Thrown when connection problems occur.

7.7 HttpHeader Class

The **HttpHeader** class is composed of two **constructors** only (constructors are not being defined in the Java Classes and Methods chapter, except where only constructors are defined). Both constructors, each named HttpHeader, construct an HttpHeader.

The **HttpHeader** class is defined as:

```
public class HttpHeader()
```

The **HttpHeader** class constructors include:

HttpHeader on page 112

HttpHeader on page 113

HttpHeader

Description

HttpHeader constructs an HttpHeader.

Syntax

```
public HttpHeader()
```

Parameters

None.

Returns

None.

Throws

None.

HttpHeader

Description

HttpHeader constructs an HttpHeader.

Syntax

```
public HttpHeader(java.lang.String name, java.lang.String value)
```

Parameters

Name	Type	Description
name	java.lang.String	The name portion of the HttpHeader.
value	java.lang.String	The value portion of the HttoHeader.

Returns

None.

Throws

None.

7.8 HttpResult Class

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

The **HttpResult** class is defined as:

```
public class HttpResult
```

The **HttpResult** class methods include:

getBinaryResult

Description

getBinaryResult gets the binary result returned from the server.

Syntax

```
public byte[] getBinaryResult()
```

Parameters

None.

Returns

Byte Array

The HTTP binary result returned from the server.

Throws

java.lang.Exception

getHeader

Description

getHeader gets a header from the list of headers.

Syntax

```
public HttpHeader getHeader(int index)
```

Parameters

Name	Type	Description
index	integer	The index of the header to retrieve.

Returns

Object

The HTTP header.

Throws

java.lang.Exception

getHeaderCount

Description

getHeaderCount gets a count of the headers.

Syntax

```
public int getHeaderCount()
```

Parameters

None.

Returns

Integer

The number of result headers from the server.

Throws

com.stc.jcsre.MarshalException: Thrown if unable to marshall contents of this object into a byte array.

getIsTextResult

Description

getIsTextResult checks the data type returned from the server; if the data is *text*, then **true** is returned; otherwise, **false** is retuned for *binary* data.

Syntax

```
public boolean getTextResult()
```

Parameters

None.

Returns

Boolean

If data received from the server is text data, **true** is returned; otherwise, if binary data, **false** is returned.

Throws

java.lang.Exception

getResponseCode

Description

getResponseCode gets the response code returned from the server. For example, 200 (HTTP_OK).

Syntax

```
public int getResponseCode()
```

Parameters

None.

Returns

Integer

The HTTP response code from the server.

Throws

java.lang.Exception

getResponseMessage

Description

getResponseMessage gets the response message returned from the server.

Syntax

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

Parameters

None.

Returns

java.lang.String

The HTTP response message from the server.

Throws

java.lang.Exception

getTextResult

Description

getTextResult gets the text result returned from the server.

Syntax

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

Parameters

None.

Returns

java.lang.String

The HTTP result from the server.

Throws

java.lang.Exception

setBinaryResult

Description

setBinaryResult sets the binary result returned from the server.

Syntax

```
public void setBinaryResult(byte[] binaryResult)
```

Parameters

Name	Type	Description
binaryResult	byte array	The HTTP binary result from the server.

Returns

None.

Throws

None.

setHeaders

Description

setHeaders sets the result headers returned from the server.

Syntax

```
public void setHeaders(java.util.Vector headers)
```

Parameters

Name	Type	Description
headers	java.util.Vector	HTTP result headers from the server in a Vector.

Returns

None.

Throws

None.

setIsTextResult

Description

setIsTextResult sets the response message returned from the server as *text* if given **true**; otherwise, sets the response message as *binary* if given **false**.

Syntax

```
public void setIsTextResult(boolean isTextData)
```

Parameters

Name	Type	Description
isTextData	boolean	The flag to indicate text data if set to *true*; otherwise, binary data is set to *false*.

Returns

None.

Throws

None.

setResponseCode

Description

setResponseCode sets the response code returned from the server.

Syntax

```
public void setResponseCode(int responseCode)
```

Parameters

Name	Type	Description
responseCode	integer	The HTTP response code from the server.

Returns

None.

Throws

None.

setResponseMessage

Description

setResponseMessage sets the response message returned from the server.

Syntax

```
public void setResponseMessage(java.lang.String responseMessage)
```

Parameters

Name	Type	Description
setResponseMessage	java.lang.String	The HTTP response message from the server.

Returns

None.

Throws

None.

setTextResult

Description

setTextResult sets the binary result returned from the server.

Syntax

```
public void setTextResult(java.lang.String textResult)
```

Parameters

Name	Type	Description
textResult	java.lang.String	The HTTP text result from the server.

Returns

None.

Throws

com.stc.jcsre.UnmarshalException: Thrown if unable to interpret the blob into the internal class attributes.

7.9 HttpsSecurityProperties Class

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

The **HttpsSecurityProperties** class is defined as:

```
public final class HttpsSecurityProperties
```

The **HttpsSecurityProperties** class methods include:

addProvider on page 120
getKeyManagerAlgorithm on page 120
insertProviderAt on page 123
setKeyManagerAlgorithm on page 123

addProvider

Description

addProvider adds a Cryptographic Service Provider (provider). This method adds a JSSE provider implementation to the list of provider implementations. This method must be called, after calling **HttpsSystemProperties.setHttpsImplementatio**n, in order to use HTTP(S).

Syntax

```
public static void addProvider(java.lang.String providerClass)
```

Parameters

Name	Type	Description
providerClass	java.lang.String	The JSSE Cryptographic Service Provider to add to the list of JSSE provider implementations.

Returns

None.

Throws

java.lang.Exception

getKeyManagerAlgorithm

Description

getKeyManagerAlgorithm gets the default Key Manager Algorithm name previously set.

Syntax

```
public static java.lang.String getKeyManagerAlgorithm()
```

Parameters

None.

Returns

java.lang.String

The name of the key manager algorithm that was previously set.

Throws

None.

getProviders

Description

getProviders gets a list of Cryptographic Service Providers (providers). This method retrieves a list of the providers that were previously set.

Syntax

```
public static java.security.Provider[] getProviders()
```

Parameters

None.

Returns

Object

An array of Providers that were added or installed.

Throws

java.lang.Exception: Thrown when any generic error occurs.

getSSLServerSocketFactoryImpl

Description

getSSLServerSocketFactoryImpl gets the default SSL Server Socket Factory implementation.

Syntax

```
public static java.lang.String getSSLServerSocketFactoryImpl()
```

Parameters

None.

Returns

java.lang.String

The implementation class of SSL Server Socket Factory.

Throws

None.

getSSLSocketFactoryImpl

Description

getSSLSocketFactoryImpl gets the default SSL Socket Factory implementation.

Syntax

```
public static java.lang.String getSSLSocketFactoryImpl()
```

Parameters

None.

Returns

java.lang.String

The implementation class of SSL Socket Factory.

Throws

None.

getTrustManagerAlgorithm

Description

getTrustManagerAlgorithm gets the default trust manager algorithm name previously set.

Syntax

```
public static java.lang.String getTrustManagerAlgorithm()
```

Parameters

None.

Returns

java.lang.String

The name of the trust manager algorithm that was previously set.

Throws

None.

getX509CertificateImpl

Description

getX509CertificateImpl gets the X509Certificate implementation.

Syntax

```
public static java.lang.String getX509CertificateImpl()
```

Parameters

None.

Returns

java.lang.String

The implementation class of the X509Certificate.

Throws

None.

insertProviderAt

Description

insertProviderAt adds a Cryptographic Service Provider (provider). This method adds a JSSE provider implementation to the list of provider implementations. This method must be called after calling **HttpsSystemProperties.setHttpsImplementation**, in order to use HTTP(S). The provider is inserted into the list at the position indicated by **position**.

Syntax

```
public static void insertProviderAt(java.lang.String providerClass,
    int position)
```

Parameters

Name	Type	Description
providerClass	java.lang.String	The JSSE Cryptographic Service Provider to add to the list of JSSE provider implementations.
position	integer	The position to insert this Provider.

Returns

None.

Throws

java.lang.Exception

setKeyManagerAlgorithm

Description

setKeyManagerAlgorithm sets the default Key Manager Algorithm name.

Syntax

```
public static void setKeyManagerAlgorithm(java.lang.String
    keyManagerAlgoName)
```

Parameters

Name	Type	Description
keyManagerAlgoName	java.lang.String	The name of the key manager algorithm to use.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default key manager algorithm name.

setSSLServerSocketFactoryImpl

Description

setSSLServerSocketFactoryImpl sets the default SSL Socket Factory implementation.

Syntax

```
public static void setSSLServerSocketFactoryImpl
    (java.lang.String sslServerSocketFactoryImplClass)
```

Parameters

Name	Type	Description
sslServerSocketFactoryImplClass	java.lang.String	The implementation class of SSL Server Socket Factory. For example, if the implementation class is called MySSLServerSocketFactoryImpl and it appears in the com.radcrypto package, you should specify com.radcrypto.MySSLServerSocketFactoryImpl.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default SSL Server Socket Factory implementation.

setSSLSocketFactoryImpl

Description

setSSLSocketFactoryImpl sets the default SSL Socket Factory implementation.

Syntax

```
public static void setSSLSocketFactoryImpl
    (java.lang.String sslSocketFactoryImplClass))
```

Parameters

Name	Type	Description
sslSocketFactoryImplClass	java.lang.String	The implementation class of SSL Socket Factory.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default SSL Socket Factory implementation.

setTrustManagerAlgorithm

Description

setTrustManagerAlgorithm sets the default Trust Manager Algorithm name.

Syntax

```
public static void setTrustManagerAlgorithm(java.lang.String
    trustManagerAlgoName)
```

Parameters

Name	Type	Description
trustManagerAlgoName	java.lang.String	The name of the trust manager algorithm to use.

Returns

None.

Throws

java.lang.String: Thrown if unable to set the default trust manager algorithm name.

setX509CertificateImpl

Description

setX509CertificateImpl sets the X509Certificate implementation.

Syntax

```
public static void setX509CertificateImpl(java.lang.String
    x509CertificateImpl)
```

Parameters

Name	Type	Description
x509CertificateImpl	java.lang.String	The implementation class of X509Certificate.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the X509Certificate implementation.

7.10 HttpsSystemProperties Class

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

The **HttpsSystemProperties** class is defined as:

```
public final class HttpsSystemProperties
```

The **HttpsSystemProperties** class methods include:

getHttpsProtocolImpl

Description

getHttpsProtocolImpl gets the HTTP(S) protocol implementation that was previously set.

Syntax

```
public static java.lang.String getHttpsProtocolImpl()
```

Parameters

None.

Returns

java.lang.String

The HTTP(S) protocol implementation package name. If the name was not previously set, a null is returned.

Throws

None.

---

getKeyStore

Description

getKeyStore gets the default KeyStore previously set.

Syntax

```
public static java.lang.String getKeyStore()
```

Parameters

None.

Returns

java.lang.String

The full path file name specifying the KeyStore if previously set; otherwise, returns a null.

Throws

None.

---

getKeyStorePassword

Description

getKeyStorePassword gets the default KeyStore password previously set.

Syntax

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

Parameters

None.

Returns

java.lang.String

The KeyStore password that was previously set; returns a null if not previously set.

Throws

None.

getKeyStoreType

Description

getKeyStoreType gets the default KeyStore type previously set.

Syntax

```
public static java.lang.String getKeyStoreType()
```

Parameters

None.

Returns

java.lang.String

The KeyStore type that was previously set; returns a null if not previously set.

Throws

com.stc.jcsre.MarshalException: Thrown if unable to marshall contents of this object into a byte array.

getTrustStore

Description

getTrustStore gets the default TrustStore previously set.

Syntax

```
public static java.lang.String getTrustStore()
```

Parameters

None.

Returns

java.lang.String

Full path file name specifying the TrustStore, if previously set.

Throws

None.

getTrustStorePassword

Description

getTrustStorePassword gets the default TrustStore password previously set.

Syntax

```
public static java.lang.String getTrustStorePassword()
```

Parameters

None.

Returns

java.lang.String

The TrustStore password that was previously set; returns a null if it was not previously set.

Throws

None.

getTrustStoreType

Description

getTrustStoreType gets the default TrustStore type previously set.

Syntax

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

Parameters

None.

Returns

java.lang.String

The name of the TrustStore type, if it was previously set.

Throws

None.

setHttpsProtocolImpl

Description

setHttpsProtocolImpl sets the HTTP(S) protocol implementation. This method adds the "https" **URLStreamHandler** implementation by including the handler's implementation package name to the list of packages that are searched by the Java URL class.

Syntax

```
public static void setHttpsProtocolImpl(java.lang.String
    httpsProtocolImpl)
```

Parameters

Name	Type	Description
httpsProtocolImpl	java.lang.String	The HTTP(S) protocol implementation package name.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the HTTP(S) protocol implementation.

setKeyStore

Description

setKeyStore sets the default KeyStore file. If the default KeyStore is not specified with this method, then the KeyStore managed by KeyManager is empty.

Syntax

```
public static void setKeyStore(java.lang.String keyStoreFile)
```

Parameters

Name	Type	Description
keyStoreFile	java.lang.String	Full path file name specifying the KeyStore. See Sun's keytool utility program for more detail.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default KeyStore.

setKeyStorePassword

Description

setKeyStorePassword sets the default KeyStore password. If the default KeyStore password is not set with this method, then the default KeyStore password is assumed to be "". See the Sun keytool for more detail.

Syntax

```
public static void setKeyStorePassword(java.lang.String
    keyStorePassword)
```

Parameters

Name	Type	Description
keyStorePassword	java.lang.String	The KeyStore password.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default KeyStore password.

setKeyStoreType

Description

setKeyStoreType sets the default KeyStore type. If the default KeyStore type is not set with this method, then the default KeyStore type "jks" is used.

Syntax

```
public static void setKeyStoreType(java.lang.String keyStoreType)
```

Parameters

Name	Type	Description
keyStoreType	java.lang.String	The KeyStore type.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default KeyStore type.

setTrustStore

Description

setTrustStore sets the default TrustStore. If the default TrustStore is not specified with this method, then a default TrustStore is searched for. For instance, if a TrustStore named l**ib/security/jssecacerts** is found, it is used. If it is not found not, a search for **lib/security/cacerts** is performed, and, if found, is used. Finally, if a TrustStore is still not found, then the TrustStore managed by the TrustManager is a new empty TrustStore.

Note: *See the Sun keytool utility program for more details.*

Syntax

```
public static void setTrustStore(java.lang.String trustStoreFile)
```

Parameters

Name	Type	Description
trustStoreFile	java.lang.String	Full path file name specifying the TrustStore.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default TrustStore.

setTrustStorePassword

Description

setTrustStorePassword sets the default TrustStore password. If the default TrustStore password is not set with this method, then the default TrustStore password is assumed to be "".

Note: *See the Sun keytool utility for more details.*

Syntax

```
public static void setTrustStorePassword(java.lang.String
    trustStorePassword)
```

Parameters

Name	Type	Description
trustStorePassword	java.lang.String	The TrustStore password.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default TrustStore password.

setTrustStoreType

Description

setTrustStoreType sets the default TrustStore type.

Syntax

```
public static void setTrustStoreType(java.lang.String trustStoreType)
```

Parameters

Name	Type	Description
trustStoreType	java.lang.String	The TrustStore type.

Returns

None.

Throws

java.lang.Exception: Thrown if unable to set the default KeyStore type.

7.11 QueryPair Class

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

The **QueryPair** class is defined as:

```
public class QueryPair
```

The QueryPair class extends

The **QueryPair** class methods include:

getName

Description

getName gets the name portion of the query pair.

Syntax

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

Parameters

None.

Returns

java.lang.String

The name portion of the query.

Throws

None.

getValue

Description

getValue gets the value portion of the query pair.

Syntax

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

Parameters

None.

Returns

java.lang.String

Value portion of the query.

Throws

None.

setName

Description

setName sets the name portion of the query pair.

Syntax

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

Parameters

Name	Type	Description
name	java.lang.String	The Name portion of the query.
_attribute	Attribute	The key-value pair mapping.

Returns

None.

Throws

None.

setValue

Description

setValue sets the value portion of the query pair. The value is URL-encoded.

Syntax

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

Parameters

Name	Type	Description
value	java.lang.String	The Value portion of the query.

Returns

None.

Throws

None.

toString

Description

toString returns the name/value query pair as URL-encoded strings. Overrides **toString** in class **java.lang.Object**.

Syntax

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

Parameters

None.

Returns

java.lang.String

The URL-encoded name/value pair string.

Throws

None.

7.12 QueryString Class

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

The **QueryString** class is defined as:

```
public class QueryString
```

The **QueryString** class methods include:

getCount on page 137
getQueryPair() on page 138
toString on page 140

add(QueryPair queryPair)

Description

add adds a name/value pairing.

Syntax

```
public void add(QueryPair queryPair)
```

Parameters

Name	Type	Description
queryPair	QueryPair	The query name/value pair.

Returns

None.

Throws

None.

add(java.lang.String name, java.lang.String value)

Description

add adds a name/value pairing.

Syntax

```
public void add(java.lang.String name, java.lang.String value)
```

Parameters

Name	Type	Description
name	java.lang.String	The name portion of the query pair.
value	java.lang.String	The value portion of the query pair.

Returns

None.

Throws

None.

clone

Description

clone constructs a **QueryString** by cloning itself.

Syntax

```
public java.lang.Object clone()
```

Parameters

None.

Returns

Object

The cloned **QueryString** object. It must be a **QueryString** because this method overrides the object **clone**() method.

Throws

None.

getCount

Description

getCount gets a count of the query pairs.

Syntax

```
public int getCount(_index, _attribute)
```

Parameters

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

Returns

Integer

The number of query pairs in the query.

Throws

None.

getQueryPair()

Description

getQueryPair gets the **QueryPair** placeholder object. Added to work with **.xsc** and Java collaboration editor for drag and drop. Once data is populated in the **QueryPair** place-holder object, use one of the other methods for managing query or form data information. If this method is not called, a default **QueryPair** object is used.

Syntax

```
public QueryPair getQueryPair()
```

Parameters

None.

Returns

Object

The **QueryPair** object being used as a place-holder for adding an HTTP query or form data information.

Throws

None.

getQueryPair(int index)

Description

getQueryPair gets a query pair from the list of query pairs..

Syntax

```
public QueryPair getQueryPair(int index)
```

Parameters

Name	Type	Description
index	integer	The index to the QueryPair that is to be retrieved.

Returns

Object

A query pair in the query.

Throws

None.

getQueryString

Description

getQueryString returns the string format of **QueryString**.

Syntax

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

Parameters

None.

Returns

java.lang.String

The string format of QueryString; the name and value pairs of each query is URL-encoded.

Throws

None.

setQueryPair

Description

setQueryPair sets the QueryPair placeholder object. Added to work with .**xsc** and **Java collaboration editor** for drag and drop. Once data is populated in the QueryPair placeholder object, use one of the other methods for managing query or form data information. If this method is not called, a default QueryPair object is used.

Syntax

```
public void setQueryPair(QueryPair qPair)
```

Parameters

Name	Type	Description
qpair	object	The QueryPair object to be used as a place holder for adding HTTP query or form data information.

Returns

None.

Throws

None.

toString

Description

toString returns the string format of **QueryString**. Overrides **toString** in class **java.lang.String**.

Syntax

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

Parameters

None.

Returns

java.lang.String

The string format of QueryString; the name and value pairs of each query are URL-encoded.

Throws

None.

Appendix A

Using the openssl Utility

This appendix is provides detailed information on how to use the **openssl** utility.

A.1 Using openssl: Introduction

The **openssl** utility is a free implementation of cryptographic, hashing, and public key algorithms such as 3DES, SHA1, and RSA respectively. This utility has many options including certificate signing, which **keytool** does not provide. You can download **openssl** from the following Web site:

http://www.openssl.org

Follow the build and installation instruction for **openssl**.

To learn more about SSL, and the high level aspects of cryptography, a good source of reference is a book entitled *SSL and TLS: Designing and Building Secure Systems* (by Eric Rescorla, Published by Addison Wesley Professional; ISBN: 0201615983).

A.2 Creating a Sample CA Certificate

The sample given in this section demonstrates the use of the **openssl** utility to create a CA. This generated CA is then used to sign a CSR (see **"Signing Certificates With Your Own CA" on page 142**), whether it is generated from **keytool** or **openssl**.

For testing purposes a sample CA can be generated. To avoid spending additional funds to have a commercial CA sign test certificates, a sample is generated and used to sign the test certificate.

Perform the following operations from the command line:

```
openssl req -config c:\openssl\bin\openssl.cnf -new -x509 -keyout
ca-key.pem.txt -out ca-certificate.pem.txt -days 365

Using configuration from c:\openssl\bin\openssl.cnf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
..............++++++
.....................++++++
writing new private key to 'ca-key.pem.txt'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

```
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:US
State or Province Name (full name) []:California
Locality Name (eg, city) []:Monrovia
Organization Name (eg, company) []:SeeBeyond
Organizational Unit Name (eg, section) []:Development
Common Name (eg, your websites domain name)
[]:development.seebeyond.com
Email Address []:development@seebeyond.com
```

You are prompted for information. You must enter a password and remember this password for signing certificates with the CA's private key. This command creates a private key and the corresponding certificate for the CA. The certificate is valid for 365 days starting from the date and time it was created.

The configuration file **C:\openssl\bin\openssl.cnf** is needed for the **req** command. The default **config.cnf** file is in the **openssl** package under **apps** sub-directory.

Note: *That to use this file in Windows, you must change the paths to use double back-slashes. See* ***"Windows openssl.cnf File Example" on page 144*** *for a complete* ***Config.cnf*** *file example, which is known to work in a Windows environment.*

A.3 Signing Certificates With Your Own CA

The example in this section shows how to create a CSR with **keytool** and generate a signed certificate for the CSR with the CA created in the previous section. The steps shown in this section, for generating a **KeyStore** and a CSR, were already explained under **"Creating a KeyStore in JKS Format" on page 69**.

Note: *No details are given here for the* ***keytool*** *commands. See* ***"Creating a KeyStore in JKS Format" on page 69*** *for more information.*

To create a CSR with keytool and generate a signed certificate for the CSR

1

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

Enter keystore password:  seebeyond
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 or 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):
```

2

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

3

```
openssl  x509  -req  -CA ca-certificate.pem.txt  -CAkey ca-key.pem.txt
    -in client.csr -out client.cer  -days 365  -CAcreateserial
```

This is how we create a signed certificate for the associated CSR. The option **-CAcreateserial** is needed if this is the first time the command is issued. It is used to create an initial serial number file used for tracking certificate signing. This certificate will be valid for 365 days.

4

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

Enter keystore password: seebeyond
keytool error: java.lang.Exception: Failed to establish chain from
    reply
```

You get an exception because there is no certificate chain in the client certificate so we have to import the CA's certificate into the **KeyStore** first. You can then import the client.cer itself to form a certificate chain. You need the following steps:

1

```
keytool  -import  -keystore clientkeystore  -file CA
    ca-certificate.pem.txt  -alias theCARoot

Enter keystore password:  seebeyond
Owner: EmailAddress=development@seebeyond.com,
CN=development.seebeyond.com, OU=Development, O=SeeBeyond,
L=Monrovia, ST=California, C=US
Issuer: EmailAddress=development@seebeyond.com,
CN=development.seebeyond.com,
OU=Development, O=SeeBeyond, L=Monrovia, ST=California, C=US
Serial number: 0
Valid from: Tue May 08 15:09:07 PDT 2001 until: Wed May 08 15:09:07
PDT 2002
Certificate fingerprints:
MD5:  60:73:83:A0:7C:33:28:C3:D3:A4:35:A2:1E:34:87:F0
SHA1: C6:D0:C7:93:8E:A4:08:F8:38:BB:D4:11:03:C9:E6:CB:9C:D0:72:D0
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

2

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

Enter keystore password: seebeyond
Certificate reply was installed in keystore
```

Now that we have a private key and an associating certificate chain in the **KeyStore clientkeystore**, we can use it as a **KeyStore** for client (e*Way) authentication. The only warning is that the CA certificate must be imported into the trusted certificate store of the Web server to which you will be connecting. Moreover, the Web server must be configured for client authentication (**httpd.conf** for Apache, for example).

This appendix contains the contents of the **openssl.cnf file** that can be used on Windows. Be sure to make the appropriate changes to the directories.

A.4 Windows openssl.cnf File Example

This section contains the contents of the **openssl.cnf file** that can be used on Windows. Be sure to make the appropriate changes to the directories.

```
#
# SSLeay example configuration file.
# This is mostly being used for generation of certificate requests.
#

RANDFILE = .rnd

####################################################################
[ ca ]
default_ca= CA_default# The default ca section

####################################################################
[ CA_default ]

dir      = G:\\openssl\\\bin\\demoCA# Where everything is kept
certs    = $dir\\certs    # Where the issued certs are kept
crl_dir= $dir\\crl   # Where the issued crl are kept
database= $dir\\index.txt# database index file.
new_certs_dir= $dir\\newcerts# default place for new certs.

certificate= $dir\\cacert.pem    # The CA certificate
serial   = $dir\\serial    # The current serial number
crl      = $dir\\crl.pem    # The current CRL
private_key= $dir\\private\\cakey.pem   # The private key
RANDFILE= $dir\\private\\private.rnd # private random number file

x509_extensions= x509v3_extensions# The extentions to add to the cert
default_days= 365    # how long to certify for
default_crl_days= 30# how long before next CRL
default_md= md5      # which md to use.
preserve = no        # keep passed DN ordering

# A few difference way of specifying how similar the request should
look
# For type CA, the listed attributes must be the same, and the
optional
# and supplied fields are just that :-)
```

```
policy   = policy_match

# For the CA policy
[ policy_match ]
countryName  = match
stateOrProvinceName= match
organizationName= match
organizationalUnitName= optional
commonName   = supplied
emailAddress = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName= optional
stateOrProvinceName= optional
localityName= optional
organizationName= optional
organizationalUnitName= optional
commonName   = supplied
emailAddress = optional

####################################################################
[ req ]
default_bits= 1024
default_keyfile = privkey.pem
distinguished_name= req_distinguished_name
attributes= req_attributes

[ req_distinguished_name ]
countryName  = Country Name (2 letter code)
countryName_min= 2
countryName_max= 2

stateOrProvinceName= State or Province Name (full name)

localityName = Locality Name (eg, city)

0.organizationName= Organization Name (eg, company)

organizationalUnitName= Organizational Unit Name (eg, section)

commonName   = Common Name (eg, your website's domain name)
commonName_max= 64

emailAddress = Email Address
emailAddress_max= 40

[ req_attributes ]
challengePassword= A challenge password
challengePassword_min= 4
challengePassword_max= 20

[ x509v3_extensions ]
```

Note: *The following copyright notices apply:*

Copyright © 1998-2001 The OpenSSL Project. All rights reserved.

Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of

Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/

Index