SeeBeyond™ eBusiness Integration Suite

HTTPS e*Way Intelligent Adapter User's Guide

Release 4.5.2

Java-enabled

Version 20011019122004.

Contents

Chapter 5

Chapter 6

Appendix B

Chapter 1

Introduction

This document describes how to install and configure the Java-enabled version of the HTTPS e*Way.

1.1 Overview

The Java-enabled version of the HTTPS e*Way Intelligent Adapter (HTTPS e*Way) allows integration with third party applications using HTTP protocol (Hyper-Text Transfer Protocol) and HTTPS (HyperText Transfer Protocol over SSL). This e*Way supports both the 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 is an example of 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 above sample, the name portion of the query is "p", and the value to search is "seebeyond integrator". A query may consist of one or more of these name-value pairs.

Note: *See the HTTP specification for more details.*

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 of 255 or 1024 maximum number of characters (depending on the web server) that the GET method has. As with GET, the web page must support the POST method in order to use POST. Taking the above URL as an example, the user specifies *http://google.yahoo.com/bin/query* as the URL, and then specifies the name value pair separately. The HTTP Client allows for specification of the URL and n-number of value pairs through its methods.

The HTTP/HTTPS Java-enabled e*Way also supports automatic URL redirection. Automatic redirection occurs when the e*Way receives a 300 status code, specifically 301.

The HTTPS 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 will be used and described in this document.

Cookies are also supported. Essentially a cookie is an HTTP header. An HTTP header is a key-value pair that gets added to the header section of an HTTP message. The HTTP message has basically two parts: the header and the body. A sample header appears below:

```
HTTP 1.1 200 OK
Date: Mon, 18 Oct 1999 20:06:48 GMT
Server: Apache/1.3.4 (Unix) PHP/3.0.6 mod_perl/1.17
Last-Modified: Mon, 18 Oct 1999 12:58:21 GMT
ETag: “1e05f2-89bb-380b196d”
Accept-Ranges: bytes
Content-Length: 35259
Connection: close
Content-Type: text/html
```

The header consists of the HTTP version (HTTP 1.1) and the status code, along with the status message (200 OK). Following the HTTP version and status are eight headers, Date, Server, Last-Modified, ETag, Accept-Ranges, Content-Length, Connection, and Content-Type. An application can supply any header to be sent along with the HTTP message. The most common, known headers are Accept-type, Content-length, and Content-type.

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 appears below:

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

The server requests for the client to store the 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 will extract the cookie “sessauth=44c46a10” and return 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.

An HTTP message contains the body after the header. An example would be an HTML document sent back if you specify the URL:

```
http://www.ibiblio.org/javafaq/books/jnp2e/examples/index.html

<html>
<head>
<title>
Examples from Java Network Programming, 2nd Edition
</title>
<META name="description" content="This site contains the complete
source code for all examples from Java Network Programming by Elliotte
Rusty Harold,
O'Reilly and Associates, 2000">
<META name="keywords" content="Java, source, code, network
programming,
Elliotte, Rusty, Harold, O'Reilly">
</head>
<body bgcolor=#ffffff text=#000000>

<h1>Examples from Java Network Programming, 2nd Edition</h1>
```

```
<p>
If you want any one particular program from the book,
you should be able to find it here fairly easily.
 The complete set
of examples is available for anonymous ftp
 from
<a href="ftp://metalab.unc.edu/pub/languages/java/javafaq/
jnp2examples.tar.gz">ftp://metalab.unc.edu/pub/languages/java/
javafaq/jnp2examples.tar.gz</a>.
as well as for  browsing here.
If you download this, please do me a favor and do not put this
on any web site. Once mirror copies start proliferating it becomes
impossible
to correct any mistakes that are found.
The canonical site for these exmaples is
<a href="http://ibiblio.org/javafaq/books/jnp2e/examples/">http://
ibiblio.org/javafaq/books/jnp2e/examples/</a>.
If you're reading this at any other URL, you may not have the most up-
to-date copy.
</p>

<ul>
<li>Chapter 1. Why Networked Java?
<li>Chapter 2. Basic Network Concepts
<li><a href="03/index.html">Chapter 3. Basic Web Concepts</a>
<li><a href="04/index.html">Chapter 4. Java I/O </a>
<li><a href="05/index.html">Chapter 5. Threads</a>
<li><a href="06/index.html">Chapter 6. Looking Up Internet
Addresses</a>
<li><a href="07/index.html">Chapter 7. Retrieving Data with URLs</a>
<li><a href="08/index.html">Chapter 8. HTML in Swing</a>
<li><a href="09/index.html">Chapter 9. The Network Methods of
java.applet.Applet</a>
<li><a href="10/index.html">Chapter 10. Sockets for Clients</a>
<li><a href="11/index.html">Chapter 11. Sockets for Servers</a>
<li><a href="12/index.html">Chapter 12. Secure Sockets</a>
<li><a href="13/index.html">Chapter 13. UDP Datagrams and Sockets</a>
<li><a href="14/index.html">Chapter 14. Multicast Sockets</a>
<li><a href="15/index.html">Chapter 15. The URLConnection Class</a>
<li><a href="16/index.html">Chapter 16. Protocol Handlers</a>
<li><a href="17/index.html">Chapter 17. Content Handlers</a>
<li><a href="18/index.html">Chapter 18. Remote Method Invocation</a>
<li><a href="19/index.html">Chapter 19. The JavaMail API</a>
</ul>
<HR NOSHADE SIZE="-1">
<P>
Return to <a href="../index.html">Java Network Programming, 2nd
Edition</a>
<P>


<hr>
<div align=center>
[ <A HREF="http://ibiblio.org/javafaq/">Cafe au Lait</A>
| Examples
| <A HREF="http://ibiblio.org/javafaq/books/jnp2e/corrections/
index.html">Corrections</A>
| <A HREF="http://www.amazon.com/exec/obidos/ISBN%3D1565928709/
cafeaulaitA/">Order</A>
]</div>
<hr>
Copyright 2000 <a href="http://www.macfaq.com/personal.html">Elliotte
Rusty Harold</a><br>
```

```
<a href="mailto:elharo@metalab.unc.edu">elharo@metalab.unc.edu</
a><br>
Last Modified October 9, 2000
</body>
</html>
```

1.1.1 The Java Classes that Make up the e*Way

JDK Classes

The SeeBeyond Java 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 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.

SeeBeyondClasses

There are 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
  - SSLTunnel SocketFactory
- Supporting Classes
  - QueryPair
  - QueryString

Core Classes

The core classes implement the functionality of the e*Way by exposing objects and methods to the user. HttpClientAPI is the class that implements the HTTP functionality while HttpClient wraps the HttpClientAPI to enable the SeeBeyond GUI to expose the HttpClientAPIs via the .xsc description file. HttpAuthenticator implements username 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 (i.e., "Accept-type" is the name and "text/xml" is the value). Header names are case sensitive.

The SSL related classes are used to set up the JSSE runtime environment properties. HttpsSecurityProperties is used to set the Security properties such as setting the Provider information. The HttpsSystemProperties sets the System properties relating to security, such as the KeyStore file. The SSLTunnelSocketFactory is used to create a tunnelled socket connection to the web server via a proxy.

The supporting classes are provided for convenience and further encapsulation. QueryPair encapsulates the name and value pair for part of the query. Methods are exposed for setting both the name and values without having the user 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.

1.1.2 SSL Handshake

There are two options available for setting up SSL connectivity with a web server. The first option is Server-side authentication and the second option is Dual authentication.

The majority of eCommerce web sites on the internet are configured for Server-side authentication. With Server-side authentication, the HTTPS e*Way will request a certificate from the web server and authenticate the web server by verifying that the certificate can be trusted. Essentially, the e*Way does this 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.

The other option, Dual authentication, requires both the HTTPS e*Way authentication and web server authentication. With this option, the Server-side part (web server) of the authentication process is the same as described in the previous paragraph. In addition, the web server will request a certificate from the e*Way. The e*Way will send its certificate to the web server. The web server, in turn, will authenticate the e*Way by looking into its "trust store" for a matching trusted CA certificate. So what we get is a communication channel that is established by both parties requesting for certificate information.

Figure 1 illustrates a dialog of an SSL handshake for Server-side authentication. Figure 2 illustrates an SSL handshake for Dual authentication. Figure 3 is a diagram on an HTTPS e*Way.

Figure 1 SSL Handshake for Server-Side Authentication

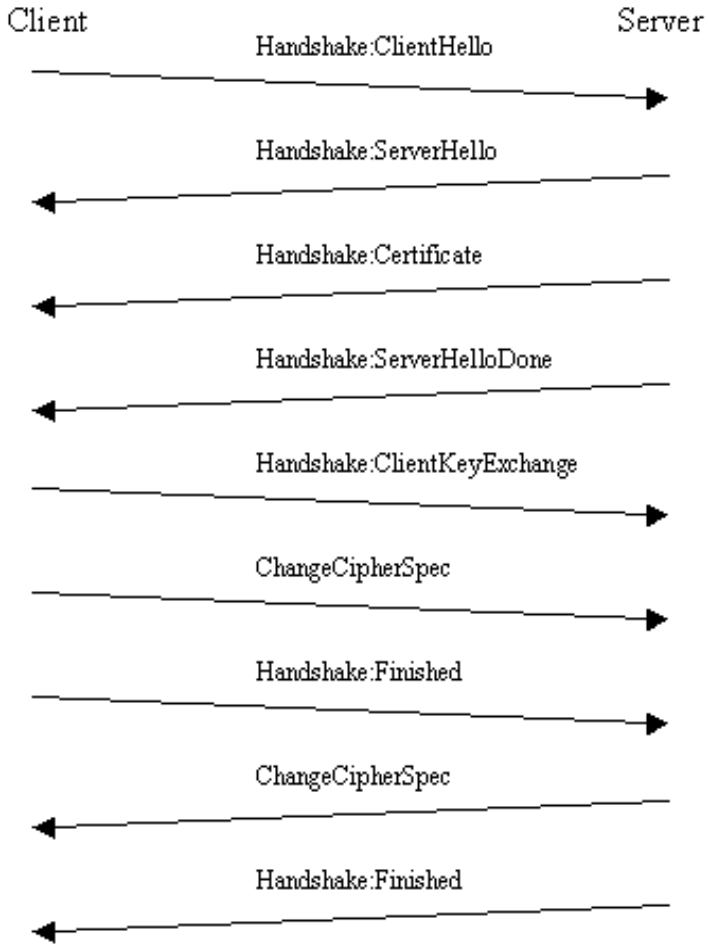


Figure 2 SSL Handshake for Dual Authentication

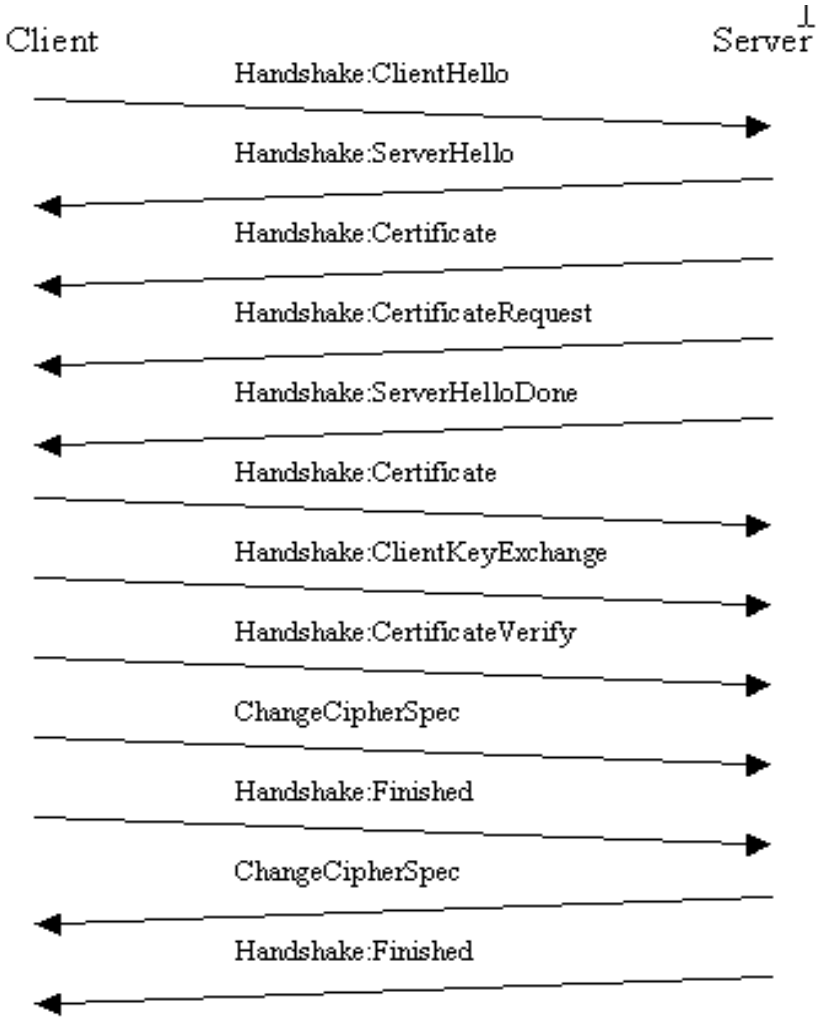
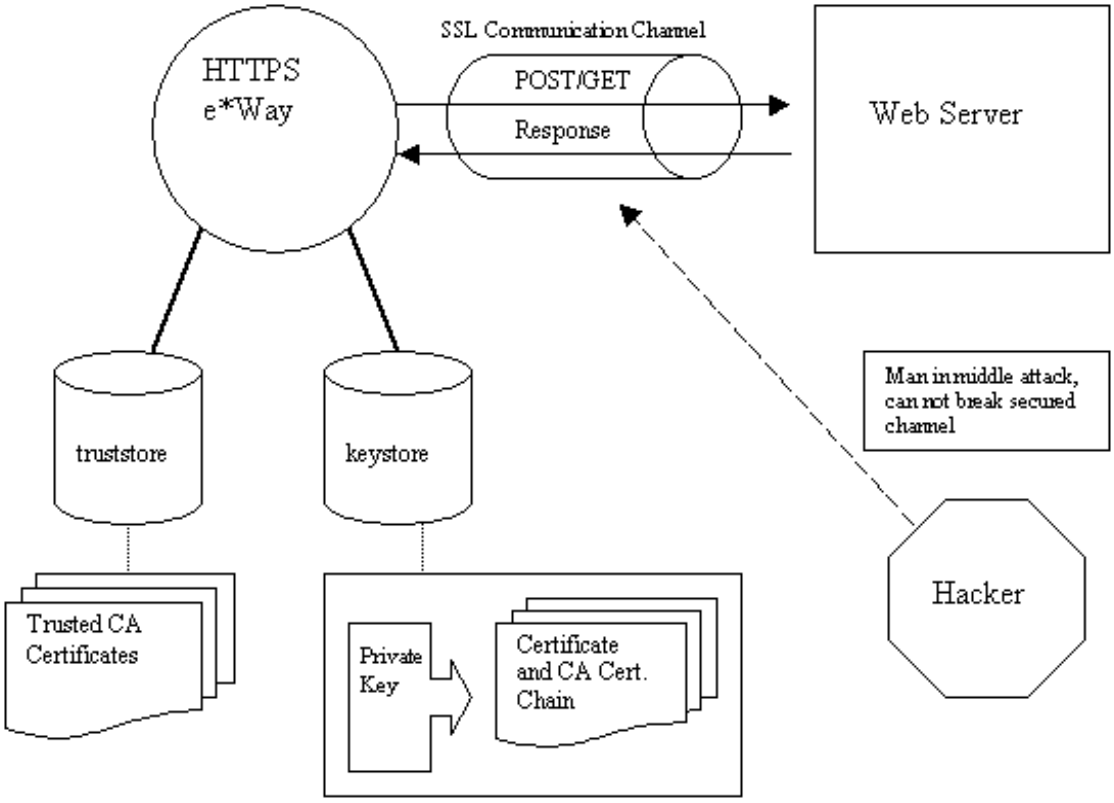


Figure 3 HTTPS e*Way



1.1.3 SSL Support

SSL is supported through the use of JSSE 1.0.2. Currently, the JSSE reference implementation is used. JSSE is a provider-based architecture. Essentially, this means that there is a set of standard interfaces for cryptographic algorithms, hashing algorithms, secured socket layered (SSL) URL stream handlers, etc. Because the user is interfacing with JSSE through these interfaces, the different components can be mixed and matched as long as the implementation is programmed under the published interfaces. However, some implementations may not support a particular algorithm. For further details, please refer to the JSSE documentation provided by Sun Microsystems at **http://java.sun.com**.

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 CA certificates. A KeyStore contains a private key, in addition to the certificate, while TrustStore only contains the certificates trusted by the client (ergo, "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 JDK 1.3 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 1.0.2 for more information.

Methods for generating a KeyStore and a TrustStore

In this section, detailed steps on how to create a KeyStore and a TrustStore (or import a certificate into an existing TrustStore such as trustedcacertsjks) will be described. The primary tool used is keytool, but openssl will be used as a reference for generating pkcs12 KeyStores. For more information on Openssl, and available downloads, see **http://www.openssl.org**.

Creating a TrustStore

For demonstration purposes, suppose we have the following CA certificates that we trust: firstCA.cert, secondCA.cert, thirdCA.cert, located in the directory C:\cascerts. We can create a TrustStore consisting of these three trusted certificates by performing three simple steps.

If you are creating a TrustStore from scratch, perform the following commands:

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

   This command 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. keytool -import -file C:\cacerts\secondCA.cert -alias secondCA -keystore myTrustStore

   This command imports the secondCA certificate into the TrustStore, myTrustStore, which was created in step 1.

Note: *This is the same command as in step 1.*

3. keytool -import -file C:\cacerts\thirdCA.cert -alias thirdCA -keystore myTrustStore

   Again, this is the same command, issued 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 29** for more information.

Using an Existing TrustStore

This section describes how to use an existing TrustStore such as trustedcacertsjks. Notice that in the previous section, "Creating a TrustStore", steps 2 and 3 were used to import two CA certificates into a TrustStore created in step 1.

For demonstration purposes, suppose we 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.

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

Once completed, trustedcacertsjks can be used as the TrustStore for the e*Way. See **"TrustStore" on page 29** for more information.

Creating a KeyStore in JKS Format

This section describes 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 request (CSR). The CA should be trusted by the server side application to which the e*Way will be connected.

Generating a KeyStore

1 keytool -keystore clientkeystore -genkey -alias client

The user is prompted for several pieces of information required to generate a Certificate signing Request (CSR). Below is a sample key generation section:

```
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=development.seebeyond.com 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 makes the key password the same as the KeyStore password).

This creates a KeyStore file clientkeystore in the current working directory. You MUST specify a fully-qualified domain for the " first and last name" question. The sample uses "***development.seebeyond.com***". The reason for this 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 Certificate Authority such as Verisign will 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.

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

This will generate 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.

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

http://www.thawte.com

or

http://www.verisign.com

See Appendix A for directions on creating your own CA with Openssl that can be used for signing certificates for development purposes.

4 If the certificate is chained with the CA's certificate, follow step A below. Otherwise, follow B. The following assumes the client certificate is in the file client.cer and the CA's certificate is in the file CARoot.cer.

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

Imports the Certificate (which may include more than one CA in addition to the Client's certificate.

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

Imports the CA's certificate into the KeyStore for chaining with the client's certificate.

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

Imports the client's certificate signed by the CA whose certificate was imported in the preceding step.

The generated file clientkeystore contains the client's private key and the associated certificate chain which is 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 29** for more information.

Creating a KeyStore in PKCS12 Format

This section describes how to create a PKCS12 KeyStore to work with JSSE. In a real working environment, a customer may 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. There are other third party tools

available for generating PKCS12 certificates. For the example below, Openssl is used to generate the PKCS12 KeyStore.

1. 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.

2. 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 29**)

   The command above 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.

1.1.4 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 expert-level knowledge of the Java Programming Language; to have expert-level knowledge of Windows and UNIX operations and administration; to be thoroughly familiar with HTTP and HTTPS protocol and to be thoroughly familiar with Windows-style GUI operations.

1.1.5 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 etc.
- **httpclient.xsc**: Allows the user to create hierarchical Event Type Definitions manually to be used in conjunction with the parsing engine contained within the extended Java Collaboration Service.
- **e*Way Connection**: The HTTP e*Way Connections provide the access to the information necessary for connection to a specified external connection.
- **Multi-mode e*Way:** Amulti-threaded host for java Collaborations.

A complete list of installed files appears in **Table 1 on page 16**.

1.2 System Requirements

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

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows 2000 (Japanese), Windows 2000 SP1 (Japanese), and Windows 2000 SP2 (Japanese)
- Windows NT 4.0 SP6a
- Windows NT 4.0 SP6a (Japanese)
- Solaris 2.6, 7, and 8
- Solaris 2.6, 7, and 8 (Japanese)
- Solaris 8 (Korean)
- AIX 4.3.3
- HP-UX 11.0 and HP-UX 11i
- HP-UX 11.0 (Japanese)
- Compaq Tru64 5.0A

To use the HTTPS e*Way, you need an e*Gate Participating Host, version 4.5 or later. For AIX operating systems, you need an e*Gate Participating Host version 4.5.1.

Chapter 2

Installation

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

2.1 Windows NT or Windows 2000

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 HTTPS e*Way on a Windows NT/ Windows 2000 system:

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 will launch. 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

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 HTTPS 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 will appear. Select the "install e*Way" option. Then follow any additional on-screen directions.

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 the Installation

The HTTP e*Way installation process will install the following files within the e*Gate directory tree. Files will be installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files created by the installation of the Java-enabled HTTPS e*Way

e*Gate Directory	File(s)
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: *Please see the HTTP e*Way Intelligent Adapter User's Guide (Monk-enabled) for information regarding the files installed by the Monk-enabled version of the e*Way).*

Chapter 3

Multi-Mode e*Way Configuration

This chapter describes how to configure the Multi-Mode e*Way.

3.1 Multi-Mode e*Way

Multi-Mode e*Way properties are set using the Enterprise Manager.

To create and configure a New 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. 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 to edit its properties.
6. When the e*Way Properties window opens, click on the **Find** button 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 on the **New** button. When the Settings page opens, set the configuration parameters for this configuration file.
8. After selecting the desired parameters, save the configuration file. Close the **.cfg** file and select **OK** to close the e*Way Properties Window.

The Multi-Mode e*Way configuration parameters are organized into the following section:

- JVM Settings

3.1.1 JVM Settings

The JVM Settings control basic Java Virtual Machine 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 O/S platforms:

OS	Java 2 JNI DLL Name
NT 4.0/ Windows 2000	jvm.dll
Solaris 2.6, 2.7, 2.8	libjvm.so
HP-UX	libjvm.sl
AIX 4.3	libjvm.a

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 (NT).

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 left unset, no paths will be 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 VM. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) will be set.

Note: *All necessary JAR and ZIP files needed by both e*Gate and the 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 will be used.

Required Values

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

Maximum Heap Size

Description

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

Required Values

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

Maximum Stack Size for Native Threads

Description

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

Required Values

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

Maximum Stack Size for JVM Threads

Description

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

Required Values

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

Class Garbage Collection

Description

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

Required Values

YES or **NO**.

Garbage Collection Activity Reporting

Description

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

Required Values

YES or **NO**.

Asynchronous Garbage Collection

Description

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

Required Values

YES or **NO**.

Report JVM Info and all Class Loads

Description

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

Required Values

YES or **NO**.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler will be 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**.

Chapter 4

e*Way Connection Configuration

This chapter describes how to configure the HTTPS e*Way Connection Configuration.

4.1 Configuring e*Way Connections

e*Way Connections are set using the Enterprise Manager.

To create and configure e*Way Connections:

1. In the Enterprise Manager's **Component** editor, select the **e*Way Connections** folder.
2. On the palette, click on the icon to create a new **e*Way Connection**.
3. The **New e*Way Connection Component** dialog box opens, enter a name for the **e*Way Connection**. Click **OK**.
4. Double-click on the new **e*Way Connection**. For this example, the connection has been defined as **SimpleHttpCP**.
5. The **e*Way Connection Properties** dialog box opens.
6. From the **e*Way Connection Type** drop-down box, 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 create a new Configuration File for this e*Way Connection. (To use an existing file, click **Find**.)

The HTTPS e*Way Connection configuration parameters are organized into the following sections:

- connector
- HTTP
- Proxies
- HttpAuthentication
- SSL

4.1.1 Connector

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.

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.

4.1.2 HTTP

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 the “SSL” section.

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,

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

AllowCookies

Description

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

Required Values

Yes or **No**.

ContentType

Description

Specifies the request content-type.

Required Values

A string. The default is set to "application/x-www-form-urlencoded". If 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" etc.

4.1.3 Proxies

The parameters in this section specify the information required for the connection point to access the external systems through a proxy server.

UseProxy

Description

Specifies whether an HTTP or HTTPS proxy will be used. If set to HTTP, then an HTTP Proxy for non-secured connection will be used. If HTTPS is selected, then an HTTPS

Proxy for secured connection will be used. Select NO if a Proxy is not used. Select AUTO to allow the e*Way to automatically switch between HTTPS proxy when using a proxy for HTTPS connections or HTTP proxy when using a proxy for HTTP connections. When AUTO is selected, both HTTPS and HTTP proxy hosts and ports must be specified. AUTO allows the e*Way to switch from an HTTPS 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

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

Required Values

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

Required Values

A valid HTTPS proxy host number.

HttpsProxyPort

Description

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

Required Values

A valid HTTPS proxy port name.

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 username is required by URLs that require "HTTP Basic Authentication" to access the site.

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

PassWord

Description

Specifies the password corresponding to the username specified previously.

Required Values

The appropriate password.

Important: *Be sure to enter a value for the **User Name** parameter before entering the **Password**.*

4.1.4 HttpAuthentication

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

UseHttpAuthentication

Description

Specifies whether standard HTTP Authentication will be used. This is used when the web site requires username and password authentication. If this 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 UseHttpAuthentication configuration parameter.

Required Values

A valid user name.

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

PassWord

Description

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

Required Values

A valid password.

Important: *Be sure to enter a value for the **User Name** parameter before entering the **Password**.*

4.1.5 SSL

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

UseSSL

Description

Specifies whether SSL needs to be configured in order to use the "https" protocol. If 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 HTTPS protocol implementation. This will add the "https" URLStreamHandler implementation by including the handler's implementation package name to the list of packages which are searched by the Java URL class. The default value specified is the package which contains the SUN reference implementation of the "https" URLStreamHandler.

Required Values

A valid package name. The default is com.sun.net.ssl.internal.www.protocol. This parameter is mandatory if using HTTPS.

Provider

Description

Specifies the Cryptographic Service Provider. This will add a JSSE provider implementation to the list of provider implementations. The default value specified is the SUN 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 using HTTPS.

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 would 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 would 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 would specify com.radcrypto.MySSLServerSocketFactoryImpl.

KeyStore

Description

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

Required Values

A valid package location.

KeyStoreType

Description

Specifies the default KeyStore type. If the default KeyStore type is not set by this method, the default KeyStore type, “jks” is used.

KeyStorePassword

Description

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

TrustStore

Description

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

Required Values

A valid TrustStore name.

TrustStore Password

Description

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

KeyManager Algorithm

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.

Chapter 5

Implementation

This chapter includes information pertinent to implementing the Java-enabled HTTPS e*Way in a production environment. Also included is a sample schema.

The following assumptions are applicable to this implementation: 1) The HTTP e*Way has been successfully installed. 2) The executable and the configuration files have been appropriately assigned. 3) All necessary .jar files are accessible.

5.1 Simple HTTP Implementation

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 Enterprise Manager GUI. To complete the implementation of the Java-enabled HTTPS e*Way, do the following:

- Make sure that the Control Broker is activated.
- In the e*Gate Enterprise Manager, define and configure the following as necessary:
  - Inbound e*Way using **stcewfile.exe**
  - Outbound e*Way using **stcewfile.exe**
  - The Multi-Mode e*Way component.
  - Event Type Definitions used to package the data to be exchanged with the external system.
  - Collaboration Rules to process Events.
  - The e*Way Connection to be created as described in **Chapter 4**.
  - Collaborations, to be associated with each e*Way component, to apply the required Collaboration Rules.
  - The destination to which data will be published prior to being sent to the external system.

The following sections describe how to define and associate each of the above components. However, the section **"Sample Schema" on page 54** provides the details necessary to create the components of a specific schema consisting of two e*Ways, three Event Types, one Collaboration Rule, two Intelligent Queues and three Collaborations.

5.1.1 Creating the New Schema

The first task in deploying the sample implementation is to create a new schema name. While it is possible to use the default schema for the sample implementation, it is recommended that you create a separate schema for testing purposes. After you install the HTTP e*Way Intelligent Adapter, do the following:

1. Start the e*Gate Enterprise Manager GUI.
2. When the Enterprise Manager prompts you to log in, select the host that you specified during installation, and enter your password.
3. You will 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 e*Gate Enterprise Manager opens under your new schema. You are now ready to begin creating the necessary components for this sample schema.

5.1.2 Event Types

The HTTPS e*Way installation includes the file "**httpclient.xsc**" which represents a standard HTTP Event Type template.

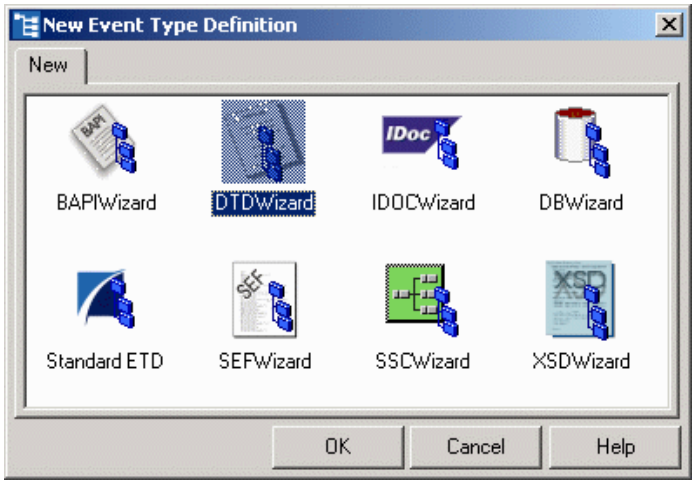
Creating an Event Type from an Existing DTD

For the purpose of this example, the following procedure shows how to create an Event Type Definition (ETD) from an existing Document Type Definition (DTD) using (Httpevent2dtd) as the input file.

1. Highlight the **Event Types** folder on the **Components** tab of the e*Gate Navigator.
2. On the palette, click the icon to create a new **Event Type**.
3. Enter the name of the **Event**, then click **OK.** (For the purpose of 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 opens.
6. Select **New** from the **File** menu on **Task Manager**.

7 The Event Type Definition Wizard opens.

Figure 4 Event Type Definition Wizard



8 Select the desired wizard. (For this Event Type, select DTDWizard.)

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.)

10 Select a DTD file to be used by the DTD builder to generate an ETD file. (Using the browse button, navigate to an existing DTD. For this sample, the file **HttpEvent2.dtd** was used.)

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

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

13 The ETD Editor opens displaying the newly converted **.xsc** file.

An ETD is a graphical representation of the layout of data in an Event.

14 Save the file as HttpEvent, and Promote the file to Run Time.

Note: *For more information on the creation and modification of Java-enabled ETDs, please see the "Java-based ETD Editor" guide.*

Creating an Event Type 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.

1 Highlight the **Event Types** folder on the **Components** tab of the e*Gate Navigator.

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

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

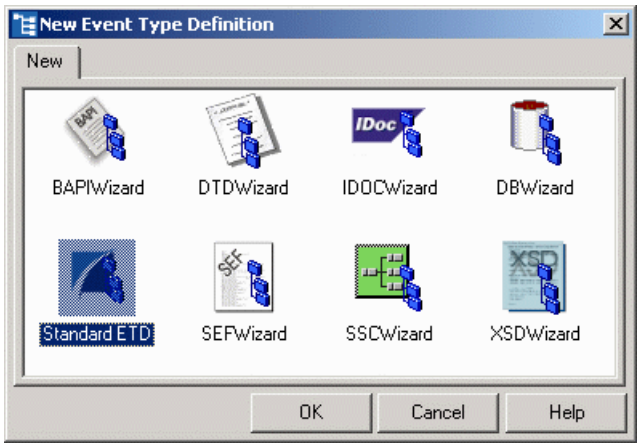
4 Select the new **Event Type**, then right-click to edit its properties.

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

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

7 The Event Type Definition Wizard opens.

Figure 5 Event Type Definition Wizard



8 Select the desired wizard. (For this Event Type, select StandardETD.)

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.)

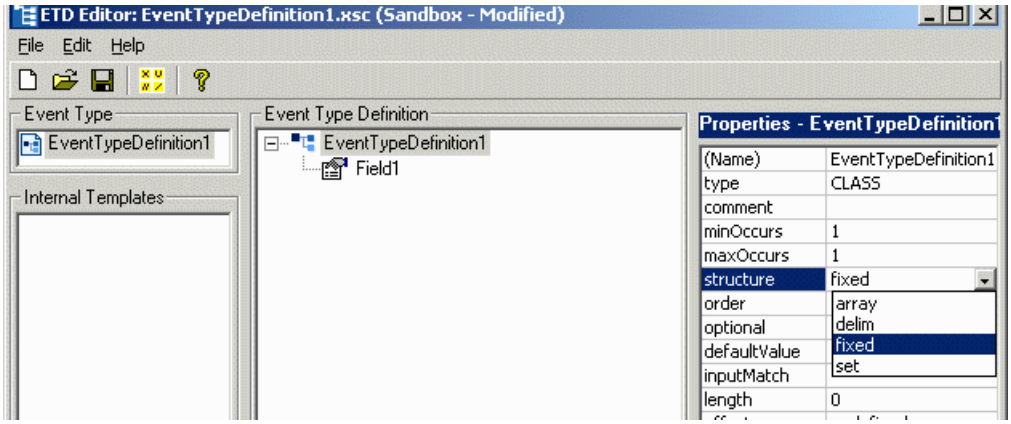
10 The ETD Editor opens, select **New** from the **File** menu.

11 Select **EventTypeDefintion1**.

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

13 Change the structure type to fixed.

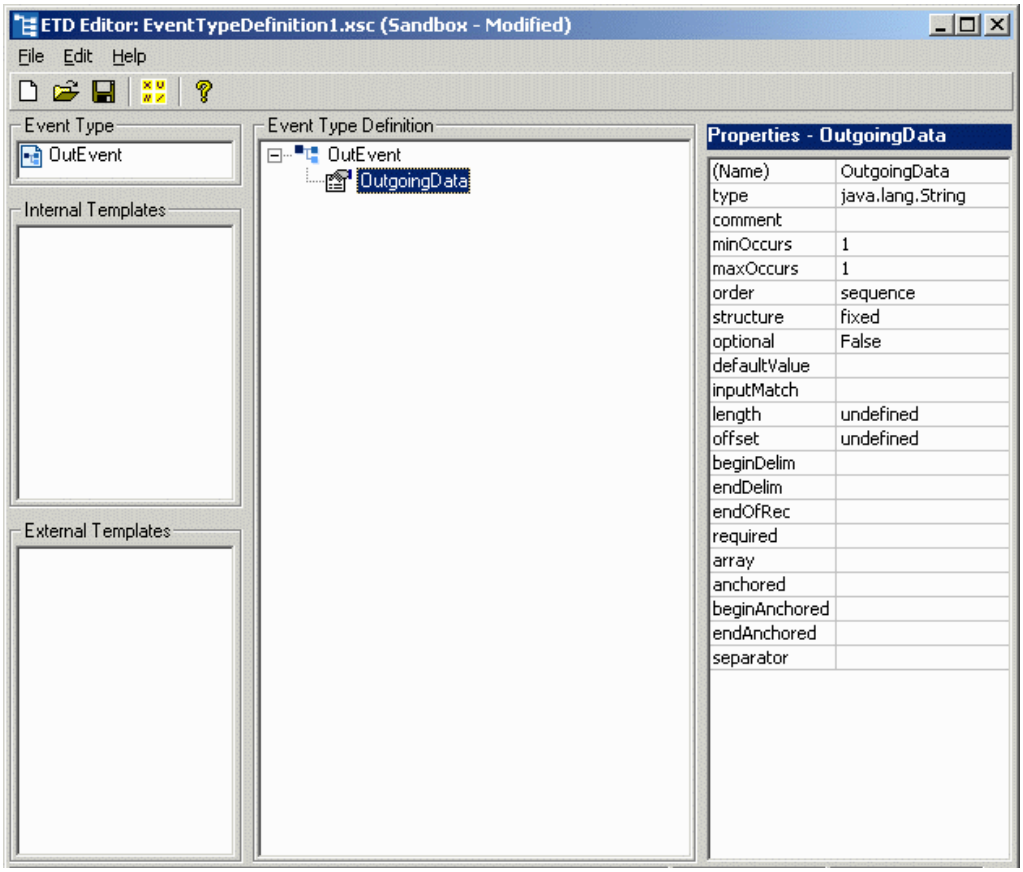
Figure 6



14 Right click on the EventTypeDefinition1, rename OutEvent.

15 Right click on the Field1, rename OutgoingData.

Figure 7



16 Save the file as OutgoingEvent.xsc, and Promote the file toRun Time.

Creating an Event Type from an Existing .xsc

For the purpose of this example, the following procedure shows how to create an Event Type Definition (ETD) from an existing .xsc file using (HttpClient.xsc) as the input file.

1. Highlight the **Event Types** folder on the **Components** tab of the e*Gate Navigator.
2. On the palette, click the icon to create a new **Event Type**.
3. Enter the name of the **Event**, then click **OK.** (For the purpose of 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** window opens, click the **Find** button.
6. Select **HttpClient.xsc** (provided as the default destination .xsc file).
7. Click OK to continue.

5.1.3 Creating and Configuring the e*Ways

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

- Inbound_eWay
- Outbound_eWay
- Multi-Mode_eWay

The following sections provide instructions for creating each e*Way.

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 will manage the new e*Ways.
4. On the palette, click the 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 window opens, 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** page opens, set the following for this configuration file:

Table 2 Configuration Parameters for the 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 the **.cfg** file.
11. Use the Startup, Advanced, and Security tabs to modify the default settings for each e*Way you configure.
    - C Use the **Startup** tab to specify whether the e*Way starts automatically, or restarts after abnormal termination or due to scheduling, etc.

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

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

12 Select **OK** to close the **e*Way Properties** window.

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 will manage the new e*Ways.

4 On the palette, click the 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 window opens, click the **Find** button beneath the **Executable File** field, and select **stcewfile.exe** for the executable file.

8 Under the **Configuration File** field, click the **New** button. When the **Settings** page opens, set the following for this configuration file:

Table 3 Configuration Parameters for the 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 Save the **.cfg** file, promote to run time and exit from **Settings**.

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 will manage 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, then click **OK**. (Http_Multi_Mode)

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

7 When the **e*Way Properties** window opens, the default **Executable File** is **stceway.exe**.

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

See **"Multi-Mode e*Way Configuration" on page 17** for details on the parameters associated with the Multi-Mode e*Way.

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

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

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

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

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

10 Select **OK** to close the **Business Object Broker Properties** window.

5.1.4 Create the e*Way Connection

The e*Way Connection configuration file contains the connection information along with the information needed to communicate via HTTPS.

To create and configure a New e*Way Connection :

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

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 sample, 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/HTTPS from the e*Way Connection Type drop-down menu, .

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

7 The e*Way Connection editor opens, select the necessary parameters.

For more information on the HTTP(S) e*Way Connection Type parameters, see **"e*Way Connection Configuration" on page 22**.

8 Save the .cfg file and Promote to Run Time.

5.1.5 Intelligent Queues

The next step is to create and associate an Intelligent Queue (IQ). IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ

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

To create and modify an Intelligent Queue for the HTTP e*Way:

1. Select the Navigator'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 icon.
6. Enter the name of the new IQ, then click **OK.** (iq_standard).
7. Select the new **IQ**, then double-click to edit its properties.
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 DLLs may be created.

   The default **Event Type Get Interval** of 100 Milliseconds is satisfactory for the purposes of this initial implementation.
9. On the **Advanced** tab, make sure that **Simple publish/subscribe** is checked under the **IQ behavior** section.

5.1.6 Collaborations Rules

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

From the **Enterprise Manager Task Bar,** select **Options** and click **Default Editor**. For the beta Release, the default should be set to **Java**.

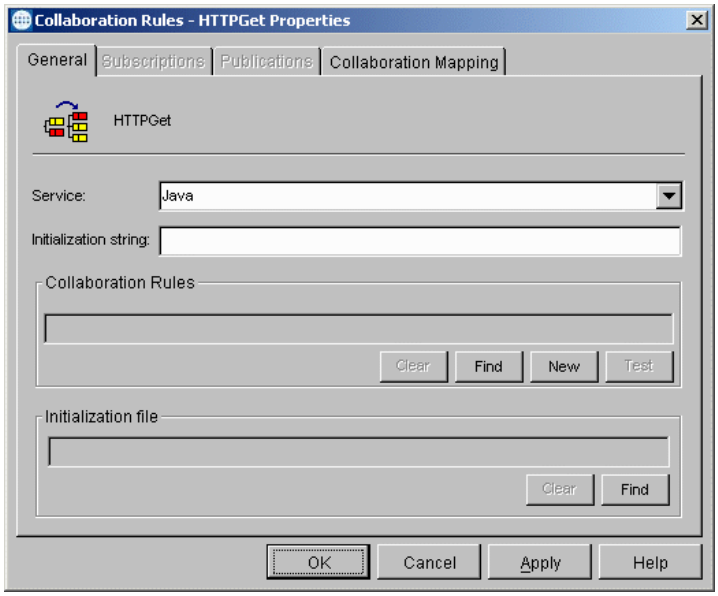
To create a Collaboration Rules file:

1. Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.
2. In the **Navigator**, select the **Collaboration Rules** folder.
3. On the palette, click the icon.
4. Enter the name of the new Collaboration Rule, then click **OK.** (Http_Get is used for the sample.)
5. Select the new **Collaboration Rule**, then right-click to edit its properties.

   The **Collaboration Rules - HTTPGet** Properties box opens.

6 On the **General** tab, in the **Service** box, select the **Java Collaboration Service**. The Collaboration Rules will use the Java Collaboration Service to manipulate Events or Event data.

Figure 8 Collaboration Rules Properties

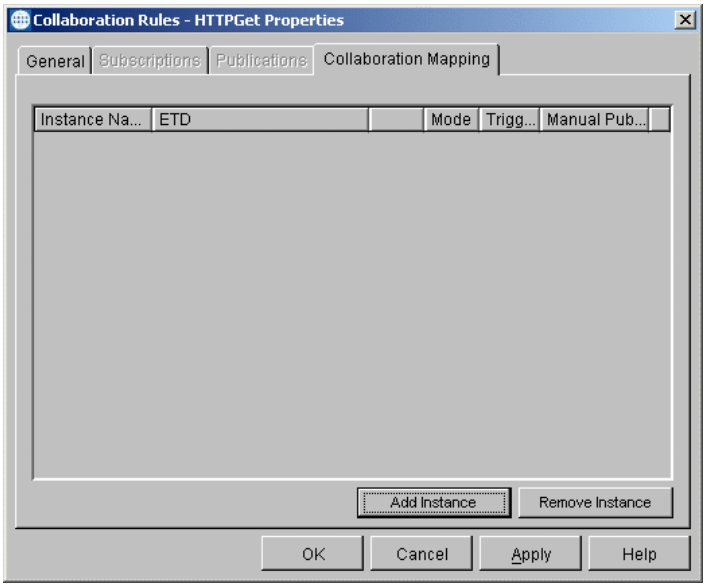


7 In the **Initialization string** box, enter any required initialization string that the Collaboration Service may require. This field can be left blank.

8 Select the **Collaboration Mapping** tab.

The Collaboration Rules - Collaboration Mapping Properties box opens:

Figure 9 Collaboration Rules - Collaboration Mapping Properties



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

For this sample, do the following:

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 9–13 using the following values:

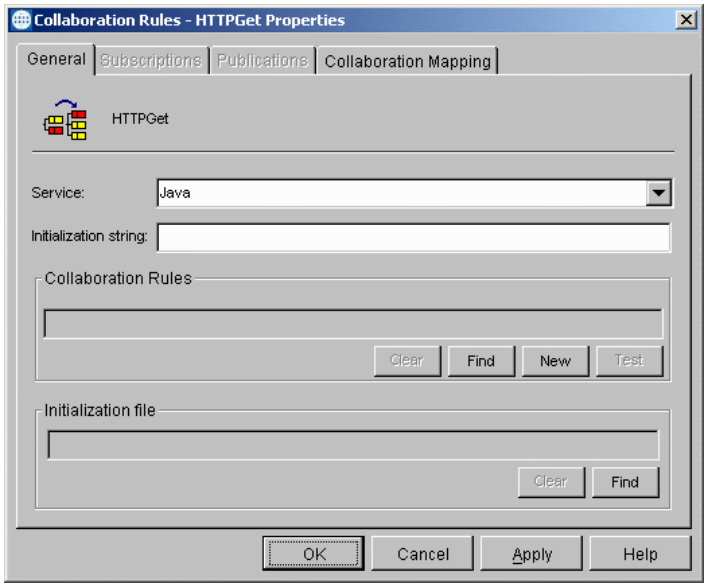
- Instance Name — **Out**
- ETD — **Outgoing_Event.xsc**
- Mode — **Out**
- Trigger — do not select.

15 Repeat steps 9 - 13 using the following values:

- Instance Name — **HttpClient**
- ETD — **HttpClient.xsc**
- Mode — **Out**
- Trigger — do not select.

16 Select the **General** tab, under the Collaboration Rule box, select **New**.

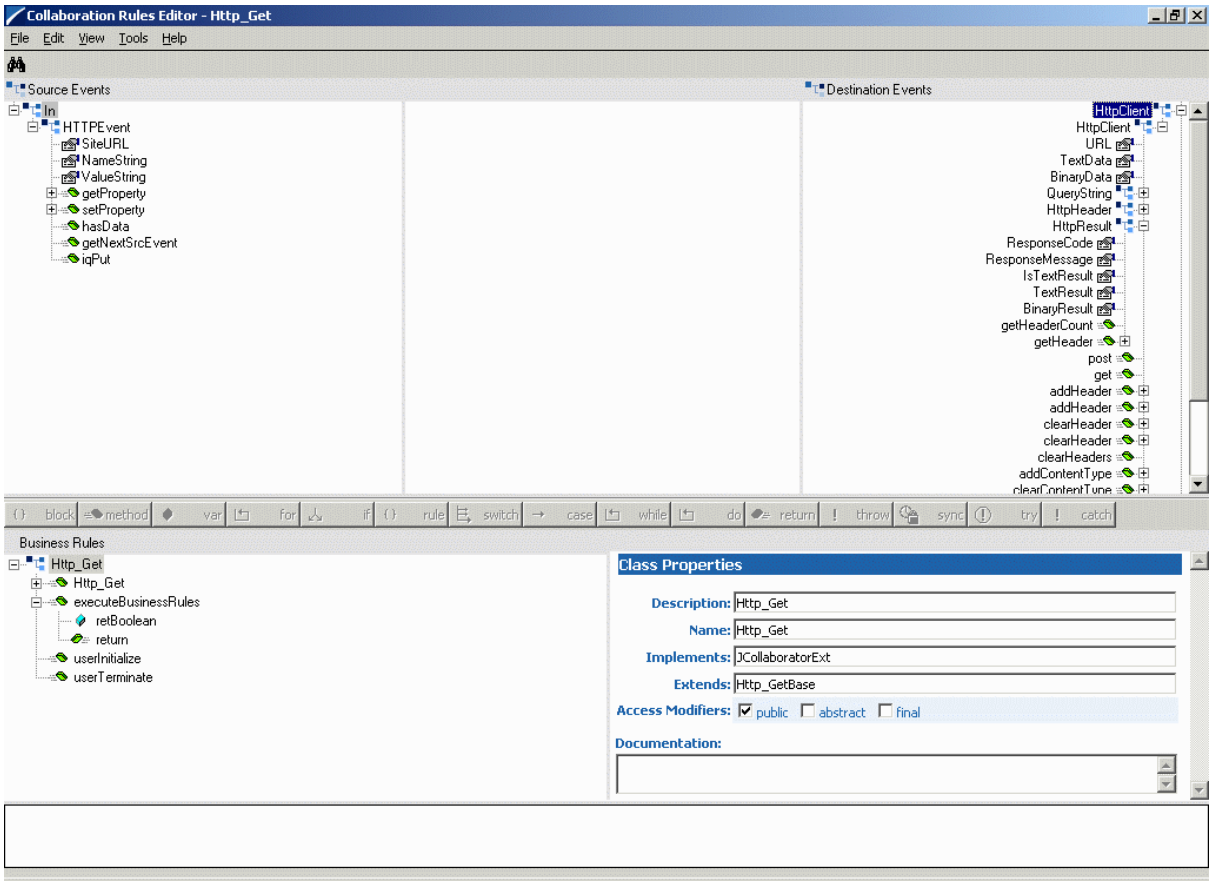
Figure 10 Collaboration Rules - Http_Get Properties General Tab



The Collaboration Rules — Collaboration Mapping Properties dialog opens.

17 Expand to full size for optimum viewing, expanding the Source and Destination Events as well.

Figure 11 Collaboration Rules — Collaboration Mapping Properties



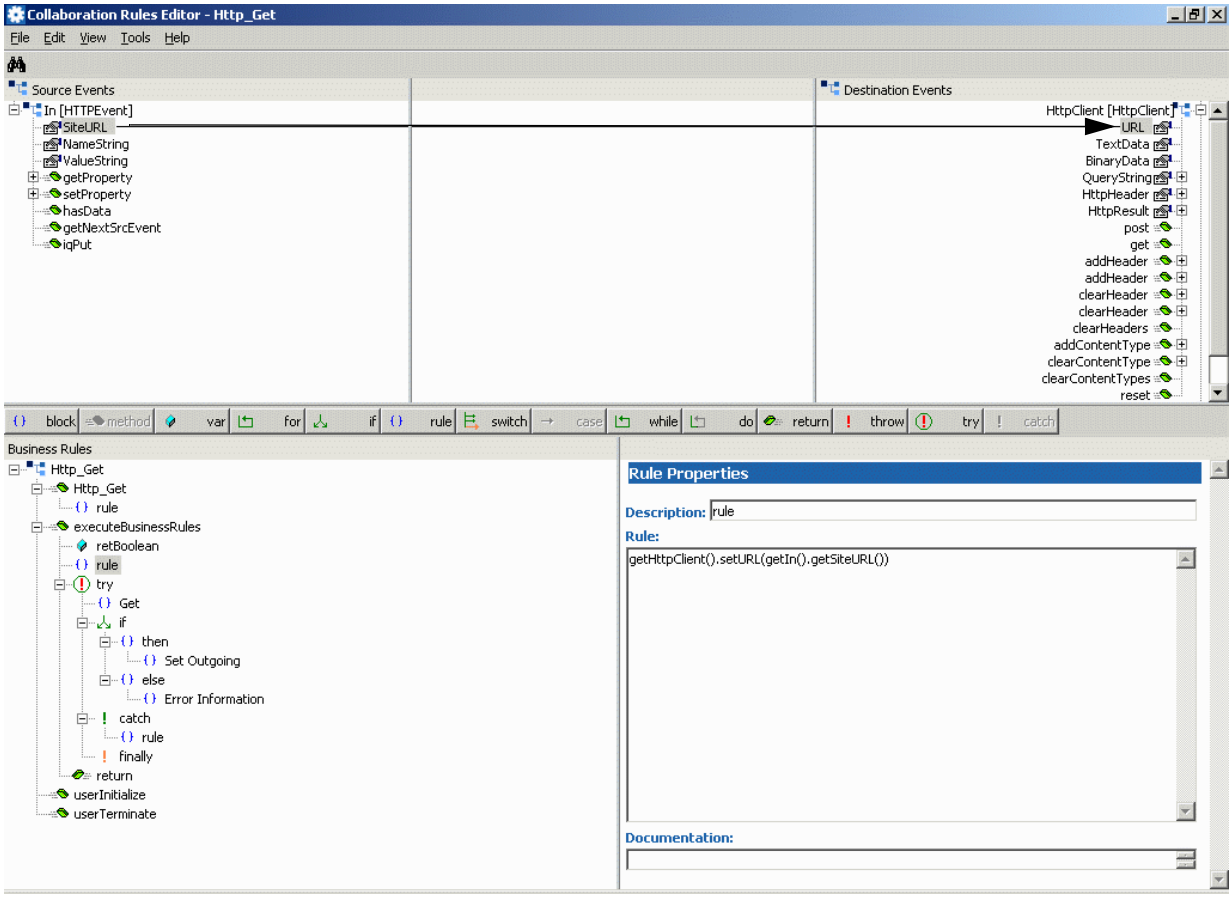
Creating the Collaboration Rules Class

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 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.

Figure 12 Collaboration Rules — Collaboration Mapping Properties



3 Select the newly created **rule** highlighted as above,

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

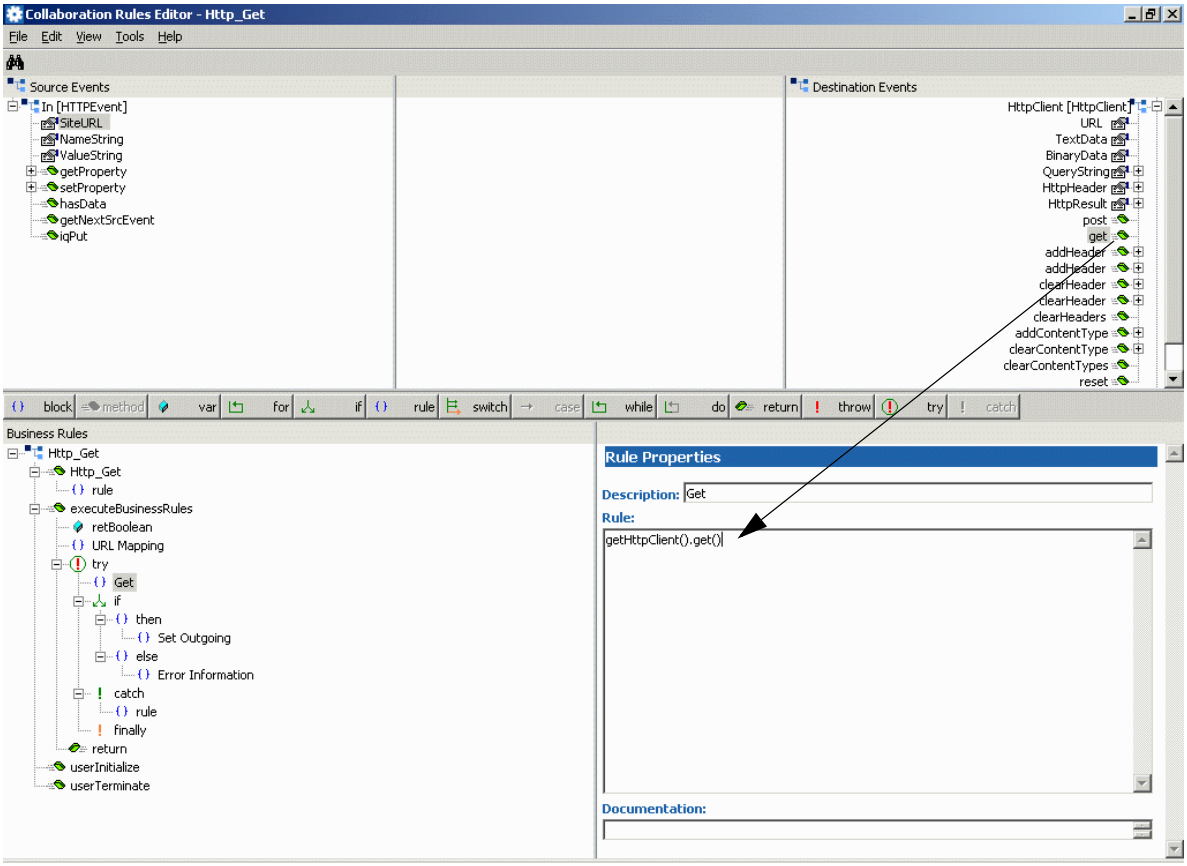
5 Click the **try** conditional expression, it will appear below the UrlMapping rule.

6 With the try conditional expression selected, click **rule**. When asked whether peer or child, select **child** (for this sample).

Figure 13 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 Rule: dialog box.

Figure 14 Collaboration Rules — Collaboration Mapping Properties



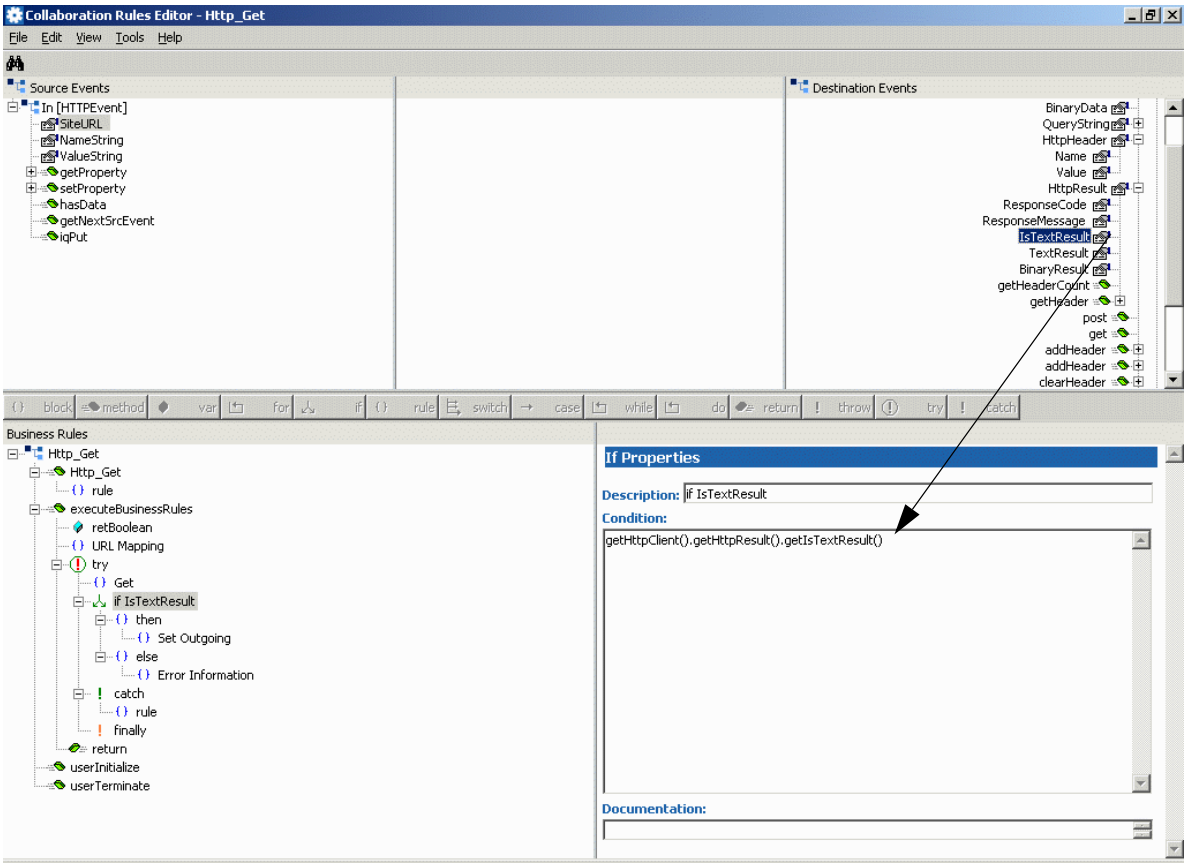
The method will appear in the Rules Properties box. Change the description from **rule** to **Get** (method).

8 With the Get method selected, click the **if** conditional expression. Drag-and-drop the **IsTextResult** to the Condition dialog box. Ensure that the condition line says:

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

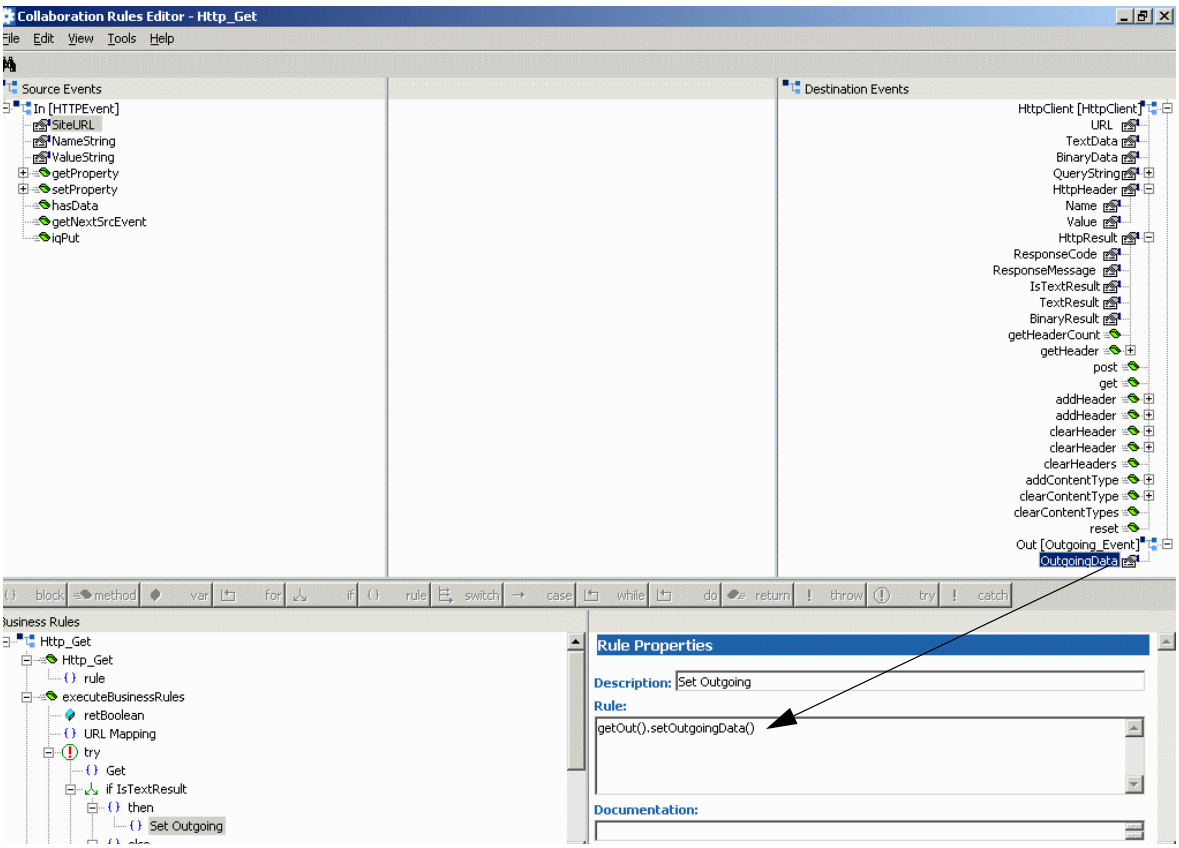
Add **IsTextResult** to the description of the conditional expression.

Figure 15 Collaboration Rules — Collaboration Mapping Properties



9 Select the **then** condition, click **rule**. Drag-and-drop the **OutgoingData** property node to the Documentation dialog box.

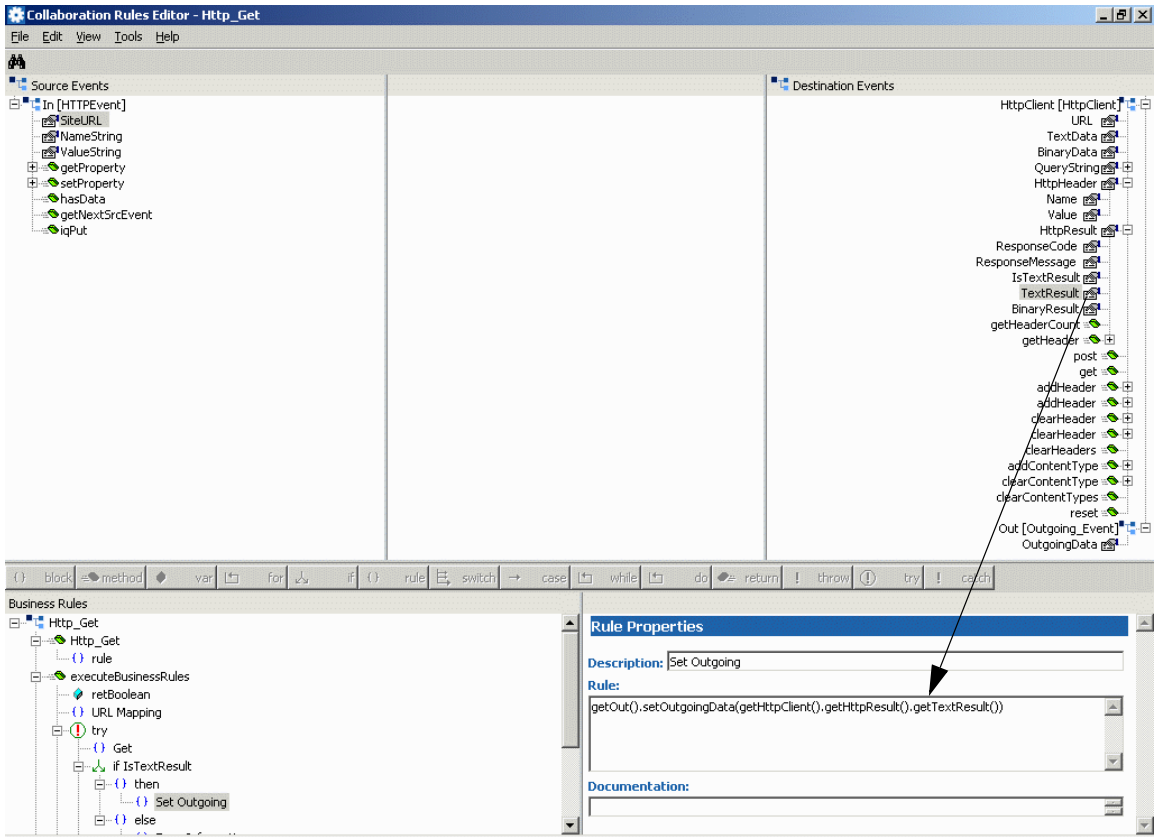
Figure 16



10 Change the Description to **SetOutgoingData.**

11 Drag-and-drop **TextResult** node into the setOutgoingData() method. Before releasing the node, ensure that the cursor is placed between the correct () (parenthesis).

Figure 17 Business Rules



12 Ensure that the condition line says:

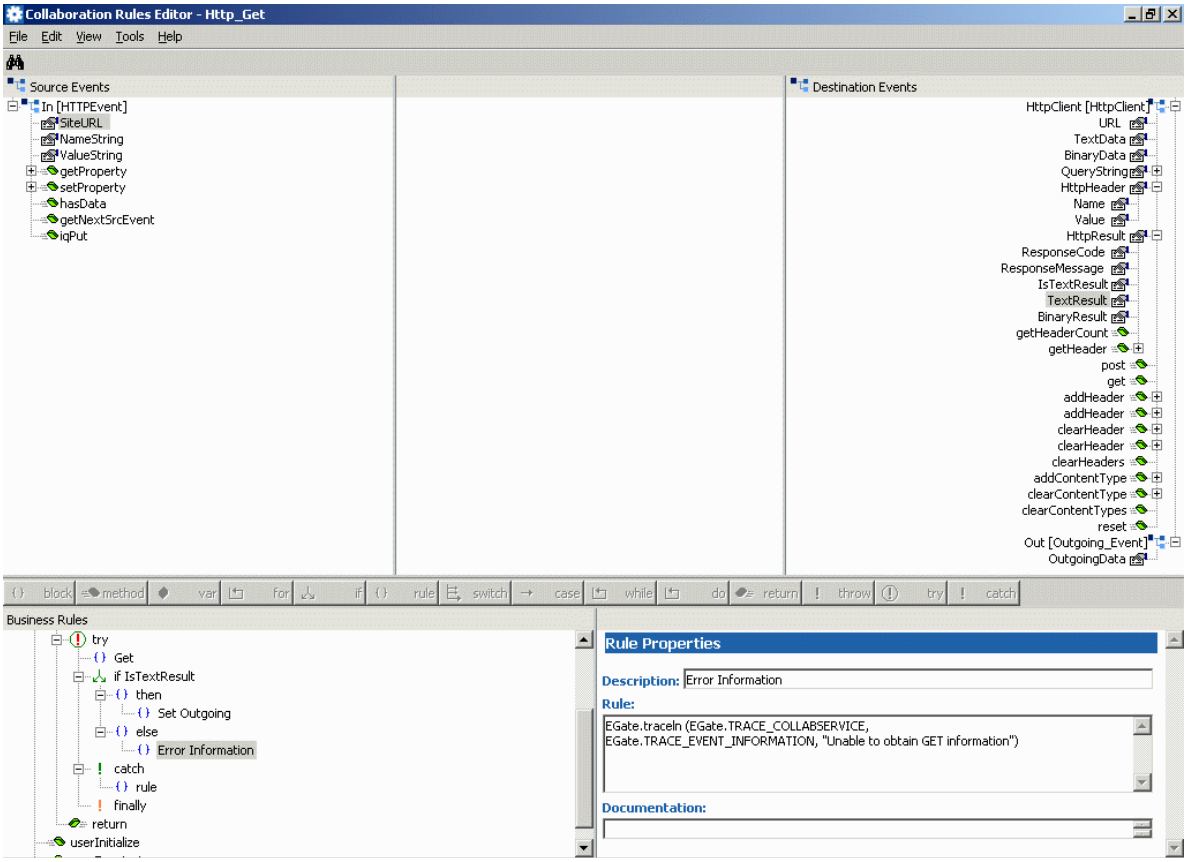
```
getOut().setOutgoingData(getHttpClient().getHttpResult().getTextResul
t())
```

13 Select the **else** condition, click **rule**.

14 Enter the desired trace information:

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

Figure 18 Business Rules

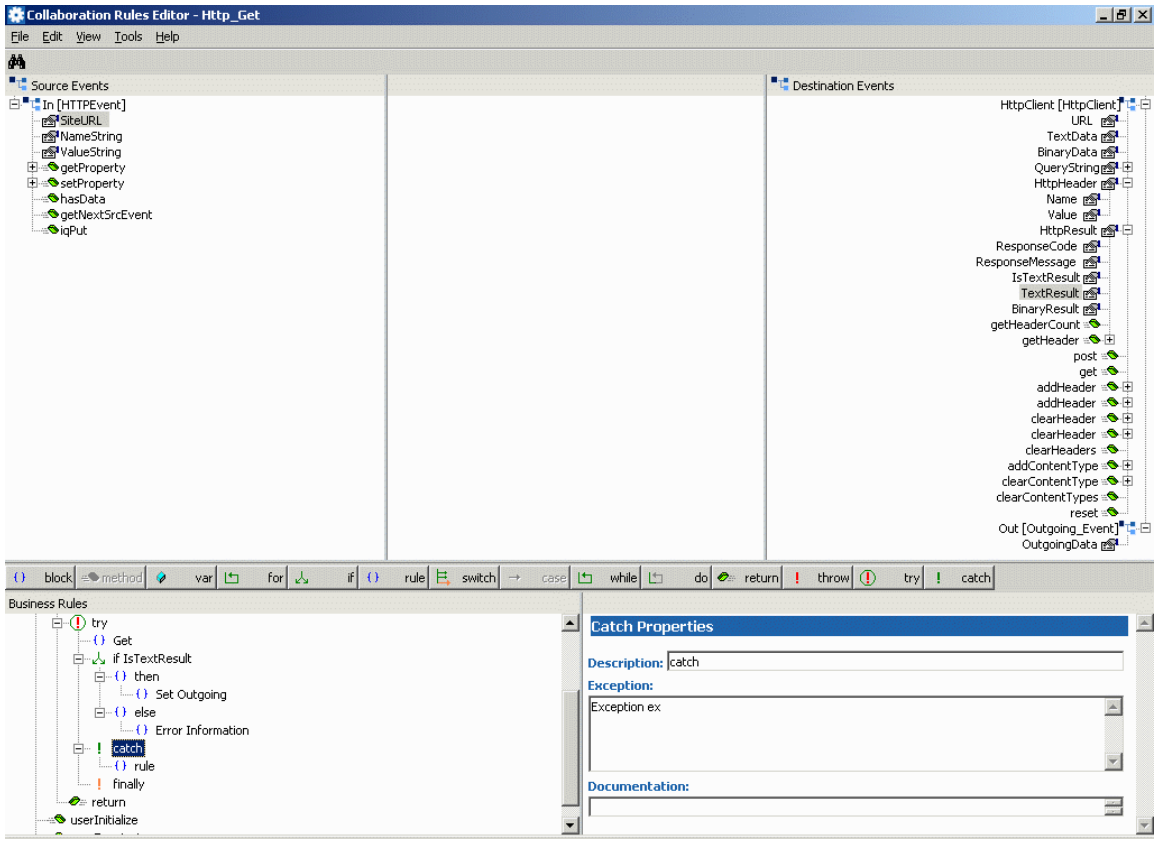


15 Select **try**, click **catch**.

16 Select **child** node. For this sample, in the Exception dialog box, type:

```
Exception ex
```

Figure 19 Business Rules



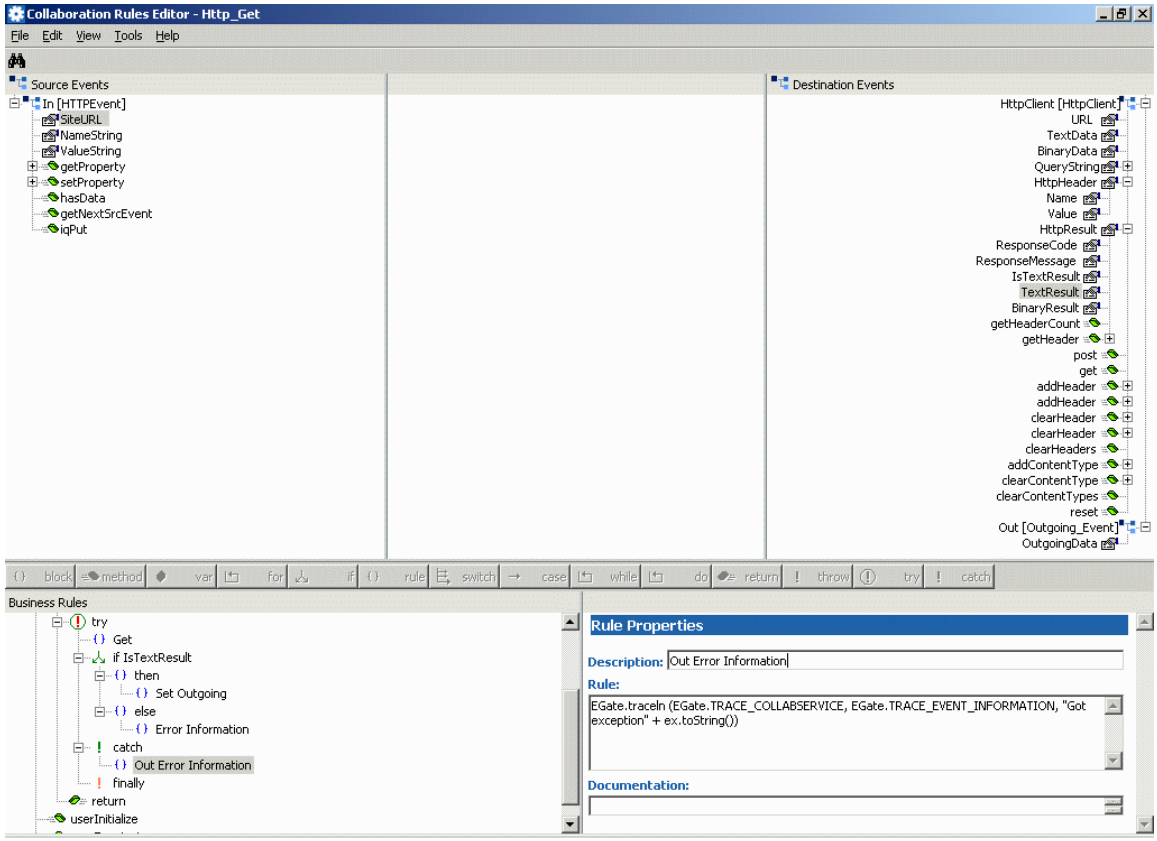
17 With **catch** selected, click **rule**.

18 In the rule dialog box type:

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

Change the name of the rule to Output Error Information.

Figure 20 Business Rules

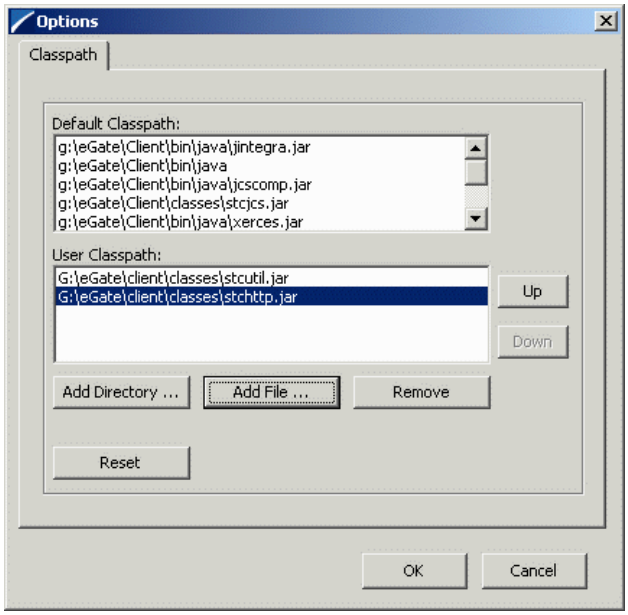


19 Before compiling the code, select **Tools**, **Options**,.

20 Verify that all necessary **.jar** files are included. For 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 21 Business Rules



21 When all the business logic has been defined, the code can be compiled by selecting **Compile** from the **File** menu. The **Save** menu opens, provide a name for the **.xpr** file. For the sample, 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.

Once the compilation is complete, save the file and exit.

22 Under the **Collaboration Rules**, the path for the .class file created appears. (For the sample, the path **"collaboration_rules\Http_Class_Get.class "**appears.)

23 Under **Initialization** file, the path for the **.ctl** file created appears. (For the sample the path **"collaboration_rules\Http_Class_Get.ctl"** appears.)

24 Click **OK** to exit the **Properties** Box.

5.1.7 Collaborations

Collaborations are the components that receive and process Event Types, then forward the output to other e*Gate components or an external. Collaborations consist of the Subscriber, which "listens" for Events of a known type (sometimes from a given source), and the Publisher, which distributes the transformed Event to a specified recipient.

Create the Inbound_eWay collaboration as follows:

1. In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.
2. Open the host on which you want to create the Collaboration.
3. Select a **Control Broker.**
4. Select the **Inbound_eWay** to assign the Collaboration.
5. On the palette, click the icon.
6. Enter the name of the new Collaboration, then click **OK.** (For the sample, "PassIn".)
7. Select the new **Collaboration**, then right-click to edit its properties.
8. From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, "PassIn".)
9. In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.
   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 will publish.
    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>.

Create the HTTP_Multi_Mode collaboration as follows:

1. In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.
2. Open the host on which you want to create the Collaboration.
3. Select a **Control Broker.**
4. Select the **HTTP_Multi_Mode** to assign the Collaboration.
5. On the palette, click the icon.
6. Enter the name of the new Collaboration, then click **OK.** (For the sample, "collab_HTTP".)
7. Select the new **Collaboration**, then right-click to edit its properties.
8. From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, "Http_Get".)
9. In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.
   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 will publish.

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 should be <SimpleHttpCP>.

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

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).

C Select the publication destination from the **Destination** list. In this case, it should be <External>.

12 Click **OK** to exit.

Create the Outbound_eWay collaboration as follows:

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

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

3 Select a **Control Broker.**

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

5 On the palette, click the icon.

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

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

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, "PassOut".)

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

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 should be <collab_HTTP>.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

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 should be <External>.

5.2 Sample Schema

The previous sections provided the basics for implementing the HTTP e*Way. This section describes how to use the HTTP e*Way within a sample Schema. The sample will send and receive Events from any platform file. It is assumed that the HTTP e*Way has been installed properly, and that all of the necessary files and scripts are located in the default location.

This implementation will consist of two file-based e*Ways, one Multi-Mode e*Way, three Event Types, three Collaboration Rules, one Intelligent Queues and three Collaboration, as follows:

- **Inbound_eWay**- This e*Way will receive input from an external source, apply pass through Collaboration Rules, and publish the information to an Intelligent Queue.
- **HTTP_Multi_Mode** - This e*Way applies extended Java Collaboration Rules to an inbound Event to perform the desired business logic, in this case a "get" from a specified URL Site.
- **Outbound_eWay -** This e*Way will receive information from the BOB and publish to the external system.
- **HttpClient**- This Event Type contains the methods to be used to perform the necessary transformation.
- **HttpEvent** - This Event Type describes an Event that is input to the extended Java Collaboration Service.
- **OutgoingEvent** - This Event Type describes an Event that contains the transformed data.
- **PassIn**- This Collaboration Rule is associated with the *Inbound_eWay*, and is used for receiving the input Event.
- **HttpGet** - The Collaboration Rule is associated with the *HTTP_BOB*, and is used to perform the transformation process.
- **PassOut** - This Collaboration Rule is associated with the Outbound_eWay, and is used for sending the Event to the External.
- **iq_standard** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the Http_bob.

This sample will perform a "get" upon a specified URL Site.

Sample Input Data

The code below is an example of input data

```
<HTTPEvent>
    <SiteURL>http://www.seebeyond.com</SiteURL>
```

```
    <MessageString></MessageString>
<HTTPEvent>
```

5.2.1 Execute the Schema

To execute the Http_Test_New schema, do the following:

1. Go to the command line prompt, and enter the following:

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

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

2. Exit from the command line prompt, and start the e*Gate Monitor GUI.
3. When prompted, specify the ***hostname*** which contains the Control Broker you started in Step 1 above.
4. Select the HTTP_Test_New schema.
5. After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, ***hostname***_igmgr, then click on the right button of the mouse, and select **Start**.
6. Highlight each of the e*Ways, right-click the mouse, and select **Start**.
7. To view the output, copy the output file (specified in the Outbound_eWay configuration file). Save to a convenient location, open.

Note: *While the schema is running, opening the destination file, will cause errors.*

Chapter 6

Java Classes and Methods

This chapter documents the Java methods that are featured in the new HTTPS e*Way. For any e*Way, of course, communication takes place both on the e*Gate system and the external side. Communication between the e*Way and the e*Gate environment is common to all e*Ways, while the communication between the e*Way the external system is different for each e*Way. For a the HTTPS e*Way, the executable file stceway.exe is used to communicate between the e*Way and the e*Gate, leaving the communiction between the e*Way and the external system open through the Java collaboration. Java methods have been added to make it easier to set information in the e*Insight Event (ETD) and to get information from it. These methods are contained in classes:

6.1 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()
```

Parameters

Name	Type	Description
httpPassWord	string	The password for HTTP authentication.

Returns

void

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	string	The password for HTTP authentication.

Returns

void

Throws

None.

setHttpUserName

Description

setHttpUserName sets the username for HTTP authentication.

Syntax

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

Parameters

Name	Type	Description
httpUserName	string	The username for HTTP authentication.

Returns

void

Throws

None.

setProxyHost

Description

setProxyHost sets the the Proxy host so that when proxy authentication is requested, this Authenticator can send the appropriate username and password to the Proxy Host..

Syntax

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

Parameters

Name	Type	Description
proxyHost	string	The host name or IP address (dotted quad address)

Returns

void

Throws

java.net.UnknownHostException - thrown 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	string	The username for Proxy authentication.

Returns

void

Throws

None.

setProxyUserName

Description

setProxyUserName sets the Proxy username for Proxy authentication.

Syntax

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

Parameters

Name	Type	Description
proxyUserName	string	The username for Proxy authentication.

Returns

void

Throws

None.

6.2 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, such that the next request sent will contain the specified value in the Content-Type header.

Syntax

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

Parameters

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

Returns

void

Throws

None.

addHeader

Description

addHeader adds the specified header, such that the next request sent will contain the specified header information.

Syntax

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

Parameters

Name	Type	Description
header	object	The header to be added.

Returns

void

Throws

None.

addHeader

Description

addHeader adds the specified header, such that the next request sent will contain the specified header information.

Syntax

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

Parameters

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

Returns

void

Throws

None.

clearContentType

Description

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

Syntax

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

Parameters

Name	Type	Description
contentType	string	The Content Type to remove.

Returns

void

Throws

None.

clearContentTypes

Description

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

Syntax

```
public void clearContentsTypes()
```

Parameters

None.

Returns

void

Throws

None.

clearHeader

Description

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

Syntax

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

Parameters

Name	Type	Description
header	object	The header to remove fromthe list of headers.

Returns

void

Throws

None.

clearHeader

Description

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

Syntax

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

Parameters

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

Returns

void

Throws

None.

---

clearHeaders

Description

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

Syntax

```
public void clearHeaders()
```

Parameters

None.

Returns

void

Throws

None.

---

get

Description

get gets the data previously set by the following methods: **setURL**, **setFormData**.

Syntax

```
public void get()
```

Parameters

None.

Returns

void

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

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

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

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

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.

Return Values

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.

Return Values

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

Throws

None.

getHttpsProxyHost

Description

getHttpsProxyHost gets the current HTTPS proxy host that was previously set.

Syntax

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

Parameters

None.

Returns

The HTTPS proxy host currently being used or null if HTTPS proxy host was not set.

Throws

None.

getHttpsProxyPort

Description

getHttpsProxyPort gets the current HTTPS proxy port that was previously set.

Syntax

```
public int getHttpsProxyPort()
```

Parameters

None.

Returns

The HTTPS proxy port currently being used or **-1** if HTTPS proxy port was not set

Throws

None.

getQueryString

Description

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

Syntax

```
public QueryString getQueryString()
```

Parameters

None.

Returns

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

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

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	string	The initialization key.
mode	integer	The mode of initialization.

Returns

void

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**, **setBinaryData**, **setFormData**. The Content-Type can be changed with **setContentType**() before posting the data.

Syntax

```
public void post()
```

Parameters

None.

Returns

void

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 memory.

Syntax

```
public boolean 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

void

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

```
void
```

Throws

None.

setHttpAuthenticator

Description

setHttpAuthenticator sets the HTTP Authenticator object for use with web sites that require username and password authentication.

Syntax

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

Parameters

Name	Type	Description
httpAuthenticator	object	The HTTP Authenticator object.

Returns

void

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

void

Throws

None.

setHttpProxyHost

Description

setHttpProxyHost sets the HTTP proxy host.

Syntax

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

Parameters

Name	Type	Description
httpProxyHost	string	The HTTP proxy host to use.

Returns

void

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

void

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

void

Throws

None.

setHttpsProxyHost

Description

setHttpsProxyHost sets the HTTPS proxy host.

Syntax

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

Parameters

Name	Type	Description
httpsProxyHost	string	The HTTPS proxy host to use.

Return Values

void

Throws

java.lang.Exception - thrown if unable to set HTTPS Proxy Host.

setHttpsProxyPort

Description

setHttpsProxyPort sets the HTTPS proxy port.

Syntax

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

Parameters

Name	Type	Description
httpsProxyPort	integer	The HTTPS proxy port to use on the HTTPS proxy host set with setHttpsProxyHost.

Return Values

void

Throws

java.lang.Exception - thrown if unable to set HTTPS Proxy Port.

setQueryString

Description

setQueryString sets the QueryString for a query.

Syntax

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

Parameters

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

Returns

void

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	string	The text data to be posted.

Returns

void

Throws

None.

setURL

Description

setURL sets the URL.

Syntax

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

Parameters

Name	Type	Description
urlString	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 hardcoded in the URL.

Returns

void

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.

6.3 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, such that the next request sent will contain the specified value in the Content-Type header.

Syntax

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

Parameters

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

Returns

void

Throws

None.

addHeader

Description

addHeader adds the specified header, such that the next request sent will contain the specified header information.

Syntax

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

Parameters

Name	Type	Description
header	HttpHeader	The header to be added.

Returns

void

Throws

None.

addHeader

Description

addHeader adds the specified header, such that the next request sent will contain the specified header information.

Syntax

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

Parameters

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

Returns

void

Throws

None.

clearContentType

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

Syntax

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

Parameters

Name	Type	Description
contentType	string	The Content Type to remove

Returns

void

Throws

None.

clearContentTypes

Description

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

Syntax

public void **clearContentTypes**( )

Parameters

None.

Returns

void

Throws

None.

clearHeader

Description

clearHeader removes, if the specified header was previously set, the header, such that the next request sent will 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

void

Throws

None.

clearHeader

Description

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

Syntax

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

Parameters

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

Returns

void

Throws

None.

clearHeaders

Description

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

Syntax

```
public void clearHeaders()
```

Parameters

None.

Returns

void

Throws

None.

---

get

Description

get gets the data previously set by the following methods: **set URL**, **setFormData**.

Syntax

```
public HttpResult get()
```

Parameters

None.

Returns

An HttpResponse structure containing the response

Throws

java.lang.Exception

java.io.IOException - thrown when an exception is returned from attempting to open an input stream fromthe 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.

---

getBinaryData

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

Syntax

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

Parameters

None.

Returns

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

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

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

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

Throws

None.

---

getHttpsProxyHost

Description

getHttpsProxyHost gets the current HTTPS proxy host that was previously set.

Syntax

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

Parameters

None.

Returns

The HTTPS proxy host currently being used, or null if HTTPS proxy host was not set.

Throws

None.

---

getHttpsProxyPort

Description

getHttpsProxyPort gets the current HTTPS proxy port that was previously set.

Syntax

```
public int getHttpsProxyPort()
```

Parameters

None.

Returns

The HTTPS proxy port currently being used, or **-1** if the HTTPS 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

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

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

The URL String previously set by setURL.

Throws

None.

post

Description

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

Syntax

```
public HttpResult post()
```

Parameters

Name	Type	Description
_index	integer	The location of the attribute.

Returns

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

void

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

void

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

void

Throws

None.

setHttpAuthenticator

Description

setHttpAuthenticator sets the HTTP Authenticator object for use with web sites that require username and password authentication.

Syntax

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

Parameters

Name	Type	Description
httpAuthenticator	object	The HTTP Authenticator object.

Returns

void

Throws

None.

setHttpProxyHost

Description

setHttpProxyHost sets the HTTP proxy host.

Syntax

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

Parameters

Name	Type	Description
httpProxyHost	string	The HTTP proxy host to use.

Returns

void

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

void

Throws

java.lang.Exception - thrown if unable to set HTTP Proxy Port.

setHttpsProxyHost

Description

setHttpsProxyHost sets the HTTPS proxy host.

Syntax

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

Parameters

Name	Type	Description
httpsProxyHost	string	The HTTPS proxy host to use.

Returns

void

Throws

java.lang.Exception - thrown if unable to set HTTPS Proxy Host.

setHttpsProxyPort

Description

setHttpsProxyPort sets the HTTPS proxy port.

Syntax

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

Parameters

Name	Type	Description
httpsProxyPort	integer	The HTTPS proxy port to use on the HTTPS proxy host set with setHttpsProxyHost.

Returns

void

Throws

java.lang.Exception - thrown if unable to set HTTPS Proxy Port.

setQueryString

setQueryString sets the QueryString for a query.

Syntax

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

Parameters

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

Returns

void

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	string	The text data to be posted.

Returns

void

Throws

None.

setURL

Description

setURL set the URL.

Syntax

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

Parameters

Name	Type	Description
urlString	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 hardcoded in the URL.

Returns

void

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.

6.4 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

void

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

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

Void

Throws

com.stc.jcsre.EBobConnectionException - thrown when connection problems occur.

6.5 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 92 **HttpHeader** on page 92

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.

6.6 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

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

The HTTP header

Throws

java.lang.Exception

getHeaderCount

Description

getHeaderCount gets a count of the headers.

Syntax

```
public int getHeaderCount()
```

Parameters

None.

Returns

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

If data received from the server is text data, the 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 esxample, 200 (HTTP_OK).

Syntax

```
public int getResponseCode()
```

Parameters

None.

Returns

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

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

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

void

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

void

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

void

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

void

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	string	The HTTP response message from the server.

Returns

void

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	string	The HTTP text result from the server.

Returns

void

Throws

com.stc.jcsre.UnmarshalException - thrown if unable to interpret the _blob into the internal class attributes.

6.7 HttpsSecurityProperties Class

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

The **HttpsSecurityProperties** class is defined as:

```
public final class HttpsSecurityProperties
```

The **HttpsSecurityProperties** class methods include:

addProvider

Description

addProvider adds a Cryptographic Service Provider (provider). This method will add a JSSE provider implementation to the list of provider implementations. This method must be called, after calling HttpsSystemProperties.setHttpsImplementation, in order to use HTTPS.

Syntax

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

Parameters

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

Returns

static void

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

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 will get a list of the providers that were previously set.

Syntax

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

Parameters

None.

Returns

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

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

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

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

the implementation class of X509Certificate.

Throws

None.

insertProviderAt

Description

insertProviderAt adds a Cryptographic Service Provider (provider). This method will add a JSSE provider implementation to the list of provider implementations. This method must be called, after calling HttpsSystemProperties.setHttpsImplementation, in order to use HTTPS. The provider will be 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	string	The JSSE Cryptographic Service Provider to add to the list of JSSE provider implementations.
position	integer	The position to insert this Provider.

Returns

static void

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	string	The name of the key manager algorithm to use.

Returns

static void

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		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

static void

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	integer	The implementation class of SSL Socket Factory.

Returns

static void

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	string	The name of the trust manager algorithm to use.

Returns

static void

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	string	The implementation class of X509Certificate.

Returns

static void

Throws

java.lang.Exception - thrown if unable to set the X509Certificate implementation.

6.8 HttpsSystemProperties Class

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

The **HttpsSystemProperties** class is defined as:

```
public final class HttpsSystemProperties
```

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

The **HttpsSystemProperties** class methods include:

getHttpsProtocolImpl

Description

getHttpsProtocolImpl gets the HTTPS protocol implementation that was previously set.

Syntax

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

Parameters

None.

Returns

static java.lang.String

The HTTPS protocol implementation package name. If not previously set, null will be returned.

Throws

None.

getKeyStore

Description

getKeyStore gets the default KeyStore previously set.

Syntax

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

Parameters

None.

Returns

Full path file name specifying the KeyStore if previously set; otherwise, returns null.

Throws

None.

getKeyStorePassword

Description

getKeyStorePassword gets the default KeyStore password previously set.

Syntax

```
public static void getKeyStorePassword()
```

Parameters

None.

Returns

The KeyStore password that was previously set. **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

The KeyStore type that was previously set. **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

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

The TrustStore password that was previously set. **Null** if not previously set.

Throws

None.

getTrustStoreType

Description

getTrustStoreType gets the default TrustStore type previously set.

Syntax

```
public void getTrustStoreType()
```

Parameters

None

Returns

static java.lang.String

Throws

None.

setHttpsProtocolImpl

Description

setHttpsProtocolImpl sets the HTTPS protocol implementation. This method will add 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	string	The HTTPS protocol implementation package name.

Returns

static void

Throws

java.lang.Exception - thrown if unable to set the HTTPS 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	string	Full path file name specifying the KeyStore. See Sun's keytool utility program for more detail.

Returns

static void

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	string	The KeyStore password.

Returns

static void

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	string	The KeyStore type.

Returns

static void

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 /lib/security/jssecacerts is found, it is used. If 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 will be a new empty TrustStore. See the Sun keytool utility program for more detail.

Syntax

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

Parameters

Name	Type	Description
trustStoreFile	string	Full path file name specifying the TrustStore.

Returns

static void

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 "". See the Sun keytool utility for more detail.

Syntax

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

Parameters

Name	Type	Description
trustStorePassword	string	The TrustStore password.

Returns

static void

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	string	The TrustStore type.

Returns

static void

Throws

java.lang.Exception - thrown if unable to set the default KeyStore type.

6.9 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

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

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	string	The Name portion of the query.
_attribute	Attribute	The key-value pair mapping.

Returns

void

Throws

None.

setValue

Description

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

Syntax

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

Parameters

Name	Type	Description
value	string	The Value portion of the query.

Returns

void

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

URL-encoded name/value pair string.

Throws

None.

6.10 QueryString Class

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

The **QueryString** class is defined as:

```
public class QueryString
```

The **QueryString** class methods include:

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

void

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	string	The name portion of the query pair.
value	string	The value portion of the query pair.

Returns

void

Throws

None.

clone

Description

clone constructs a QueryString by cloning itself.

Syntax

```
public java.lang.Object clone()
The cloned QueryString object. Needs to be casted to a QueryString as
this
    method overrides the Object clone() method.
```

Parameters

None.

Returns

The cloned QueryString object. Needs to be casted to a QueryString as this method overrides the Object clone() method.

Throws

None.

getCount

Description

getCount gets a count of the query pairs.

Syntax

```
public int getCount()
```

Parameters

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

Returns

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 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 QueryPair getQueryPair()
```

Parameters

None.

Returns

The QueryPair object being used as a placeholder for adding 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

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

The string format of QueryString; the name and value pairs of each query will be 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

void

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

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

Throws

None.

Appendix A

A.1 Openssl

The purpose of this appendix is to provide detailed information on the usage of the "openssl" utility. Openssl is a free implementation of cryptographic, hashing, and public key algorithms such as 3DES, SHA1, and RSA respectively. The openssl utility has many options including certificate signing that keytool does not provide. Openssl can be downloaded from :

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 sample follows that demonstrates the use of the openssl utility to create a CA. This generated CA is then used to sign a CSR (whether generated from keytool or openssl).

A.1.1 Creating a Sample CA Certificate

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

Perform the following from the command line.

1

```
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 will be 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 our 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" subdirectory.

Note: *That to use this file in Windows, you must change the paths to use double back-slashes. See Appendix B for a complete Config.cnf that is known to work in a Windows environment.*

A.1.2 Signing Certificates With Your Own CA

Let's create a CSR with keytool and generate a signed Certificate for the CSR with the CA we had created. The following steps for generating a KeyStore and a CSR were already described in **Creating a KeyStore in JKS Format** on page 10. No details are given here for the keytool commands; refer to the fore mentioned sections for the details.

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
```

We 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. We can then import the client.cer itself to form a certificate chain. Thus, we need two steps :

```
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

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 (eWay) authentication. The only caveat is that our CA certificate must be imported into the trusted certificate store of the web server to which we will be connecting. More over, the web server should be configured for client authentication (httpd.conf for Apache for example).

Appendix B

B.1 Openssl.cnf

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

B.1.1 Openssl.cnf for Windows

```
#
# 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 ]

# under ASN.1, the 0 bit would be encoded as 80
nsCertType   = 0x40

#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName
#nsCertSequence
#nsCertExt
#nsDataType
```

Index