CGI Web Server e*Way Intelligent Adapter User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Java Version

Copyright © 2005, 2010, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related software documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Version 20100712154945.

Contents

Chapter 1

Introduction

This chapter provides an overview of Oracle's CGI Web Server e*Way Intelligent Adapter.

1.1 Overview

The CGI Web Server e*Way acts as a gateway to the e*Gate system for a Web server. Normally, a Web server is limited to sharing local data sources only. The CGI Web Server e*Way allows the Web server to access remote data sources which would otherwise be unavailable. The CGI Web Server e*Way makes a variety of data sources available through the e*Way system.

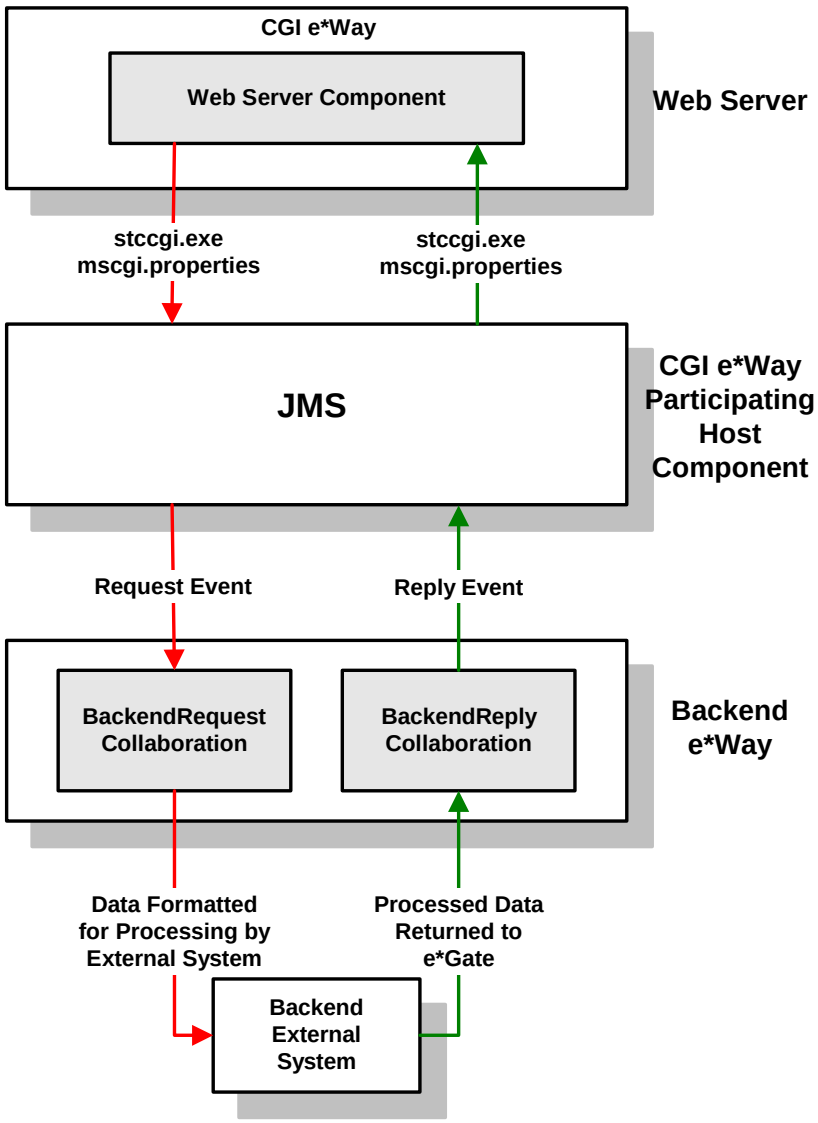
The CGI Web Server e*Way is comprised of two components, the Web server component and the Participating Host component. The Web server component resides in a Web server's CGI-bin directory. This executable, stccgi.exe, can be considered a generic CGI executable, in that it can be used with a variety of Web servers that utilize CGI. It parses CGI input supplied by a Web server using either the GET or POST method, packages the input along with the Web server's environment variables, and sends the packaged message to the Participating Host component residing on a machine where an e*Gate Participating Host is running. The Participating Host component forwards the message to the e*Gate backend system for processing. After sending the message, the Web server component waits for a reply from the Participating Host component and, upon receipt of the response, sends the response message to the Web server which can then deliver the message to the requesting Web client (for example, a browser).

For additional details, see **Chapter 4**.

1.1.1. 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 working knowledge of operations and administration for the specific operating systems under which the Web server and e*Gate run; to be familiar with CGI; and to be familiar with Windows-style GUI operations.

Figure 1 Overview of the CGI Web Server e*Way implementation



1.1.2. Components

The CGI Web Server e*Way is comprised of the following:

- **Web server components** (the CGI Web Server e*Way client) which consists of the Web server component executable and supporting library files
- **Participating Host components** (the CGI Web Server e*Way server) consisting of a Multi-mode e*Way and a JMS e*Way Connection

A complete list of installed files appears in **Table 1 on page 14**.

1.2 System Requirements

To use the CGI Web Server e*Way, you need the following:

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

The Web server that communicates with the CGI Web Server e*Way requires a client system with the following:

- A Web server, such as:
  - Apache Web Server
  - iPlanet Web Server
  - MSIIS Web Server
- Sufficient memory and disk space to support Web server functions. See the CGI Web server user's guides for more information regarding server requirements.

Note: *The e*Gate Participating Host may optionally host the Web server, but is not required to do so.*

Chapter 2

Installation

This chapter covers the installation requirements for the CGI Web Server e*Way and the Web server component configuration. A list of the files and directories created by the installation is also provided.

2.1 Installing the CGI Web Server e*Way

When installing this e*Way as part of a complete e*Gate installation, follow the instructions in the *Oracle ICAN Suite Installation Guide*. The CGI Web Server e*Way is installed as an "Add-on" component in the fourth phase of the installation. To add the CGI Web Server e*Way to an existing e*Gate installation, follow the proceeding instructions.

2.2 Windows Installation

Before installing e*Gate on your Windows system, read the following sections to ensure a smooth and error-free installation.

2.2.1. Pre-installation

- Exit all programs before running the setup program, including any anti-virus applications.
- Administrator privileges are required to install this e*Way.
- Review the **readme.txt** file provided on the installation media for important installation information.

To install the CGI Web Server e*Way on a Windows system

1. Log in as Administrator to the workstation on which the e*Way is to be installed.
2. Close any open applications.
3. Launch the setup application on the e*Gate installation CD-ROM.
4. Follow the online prompts in the InstallShield® Wizard. When the **Select Components** dialog box appears, clear all the check boxes except **Add-ons**.

5 Click **Next** as necessary to proceed through the setup application.

6 When the **User Information** dialog box appears, type your name and company name.

7 When the **Choose Destination Location** window appears, **do not** change the **Default Destination** folder unless you are directed to do so by Oracle support personnel; simply click **Next** to continue.

8 When the **Select Components** dialog box appears, select **eWays**, click the **Change.** button, and select **CGI Web Server e*Way.**

After the installation is complete, reboot the computer and launch the Schema Designer.

2.3 UNIX

Before installing the e*Way on your UNIX system, read the following sections to ensure a smooth and error-free installation.

2.3.1. Pre-installation

- Exit all programs before running the setup program, including any anti-virus applications.
- Root privileges are not required to install the e*Way.
- Review the **readme.txt** file provided on the installation media for important installation information.

To install the CGI Web Server e*Way on a UNIX system

1 Log onto the workstation on which the e*Way will be installed. If not logged in as root, you must have sufficient privileges to install files in the "egate" directory tree.

2 Insert the CD-ROM into the drive.

3 If necessary, mount the CD-ROM drive. See the *Oracle ICAN Suite Installation Guide* for information on mounting the CD-ROM on a particular UNIX systems.

4 At the shell prompt, type

cd /cdrom/setup

5 Start the installation script by typing:

setup.sh

6 If not logged in as root, a message is displayed stating that services do not start automatically for non-root users. Press **Enter** to continue.

7 A message appears to confirm that the e*Gate installation script is running, and note that typing **-** (hyphen) backs up a step, or **QUIT** (all capitals) exits the install program. Press **Enter** to continue.

8 When prompted to accept the license agreement, type **y** and press **Enter**.

9 The platform type and a menu of options is displayed:

```
Installation type (choose one):
  0. Finished with installation.  Quit.
  1. e*Gate Addon Applications
  2. e*Gate Participating Host (Client)
  3. e*Gate Registry Server
```

Type 1 to select the **e*Gate Add-on Applications** and press **Enter**.

10 When prompted for the installation path, press **Enter** to accept the default path, or enter a new path and press **Enter**.

- When logged in as root, the suggested path is **/opt/egate/client**.
- When logged in under any other user name, the suggested path is **/home/*username*/egate/client**.

Whether installing e*Gate to an application directory such as /**opt** to a **/home** directory, it is strongly recommend that the recommended relative path "egate/client" be used as the destination directory for the add-on-application installation.

11 When prompted, type **U** to update (overwrite) and press **Enter**.

Note: ***U*** *updates the installation and overwrites files as necessary.* ***M*** *creates a directory and moves everything in the current directory to* ***directoryname.old****.*

If "**U**" is selected a warning appears regarding shared EXE and DLL files. Read the warning and press **Enter** to continue.

12 Enter the name of the Registry Server that supports the add-on applications. If the installation utility detects a Registry Host running on the current server, suggests that host's name.

13 A prompt appears for the **administration login** (an e*Gate user with sufficient privilege to create components within a schema). The default is **Administrator**; unless you have created a different "administrative" user name, press **Enter** to accept the default. The default password is listed in the README.TXT file in the root directory of the installation CD-ROM.

14 Enter and confirm the password for the user specified in the step above.

Note: *e*Gate user names and passwords are case-sensitive.*

15 A menu of add-on options appears. Type the number corresponding to the desired add-on package (**CGI Web Server e*Way**) and press **Enter**.

16 Follow the on-screen instructions to complete the installation.

17 After the add-on application has been installed, the **Choose add-on packages** menu appears. Repeat step 15 to install additional packages, or **0** and press **Enter** to continue.

18 When the **Installation Type** menu appears the Add-on applications installation is complete. Do one of the following:

- To exit the setup utility, type **0** and press **Enter.**
- To continue the installation select another option .

2.4 Files/Directories Created by the Installation

The CGI Web Server e*Way installation process installs the following files within the e*Gate directory. Files are installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Server-side files installed

e*Gate Directory	File(s)
client\bin\	stccgi.exe
client\classes\	Stccgi.jar
client\ThirdParty\gnu-getopt\	gnu-getopt.jar
client\ThirdParty\jaf\jaf-1.0.1\\	activation.jar
client\ThirdParty\javamail\javamail-1.2\	mailapi.jar

Table 2 lists the files that must be manually copied onto the system running the Web server.

Table 2 Client-side files installed

Client Directory	File(s)
On IIS: \inetpub\scripts\ On Apache: \ApacheGroup\Apache\cgi-bin\ On iPlanet: \NetScape\Server4\cgi-bin\	stc_msclient.dll stc_mscommon.dll stc_msapi.dll stccgi.exe

Table 3 lists the sample files that must be copied from the Installation CD manually to the Web server scripts directory:

Table 3 Sample files :

Installation CD	File(s)
\samples\ewcgi\Java\	mscgi.properties readme.txt testmulptfrm.html testurlencoded.html webETD_CGI.zip

Sample files do NOT install automatically. They must be copied from the Installation CD to a temporary location. See **Configuring the Web Server Components** on page 23 and **The Request/Reply Sample** on page 34 for more information.

Note: *After installation, change the file permissions to allow the Web server to read and execute these files.*

Chapter 3

Configuration

3.1 Configuring the Participating Host Components in the Schema Designer

Important

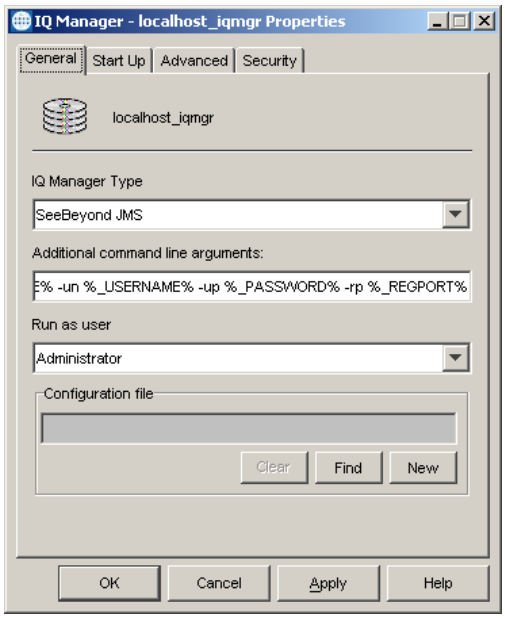
From the perspective of the e*Gate GUIs, the CGI e*Way is not a system of components distributed between the Web server and a Participating Host, but a single component that runs an executable file (the **stccgi.exe**). When this manual discusses procedures within the context of any e*Gate GUI, the term "e*Way" refers only to the Participating Host component of the CGI e*Way system.

Configuring the Participating Host components

1. If you have not already done so, launch the Schema Designer.
2. Verify that the IQ Manager Type is set to **SeeBeyond JMS**.

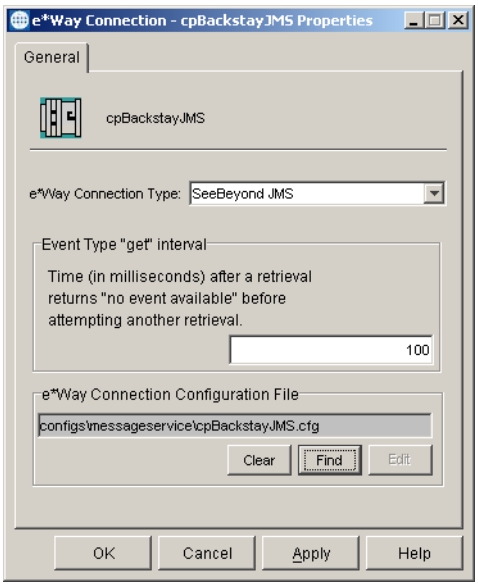
   Since the stccgi.exe publishes Events to JMS, the IQ Manager type in your Participating Host must be set to SeeBeyond JMS.

Figure 2 Oracle SeeBeyond JMS IQ Manager



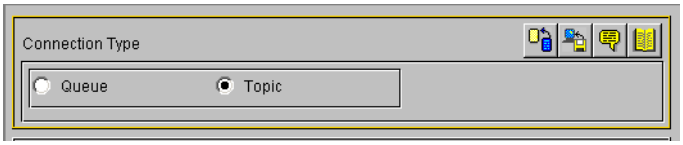
3 Create and configure an e*Way Connection. The connection type should be set to "SeeBeyond JMS". (For the sample, the e*Way Connection is referred to as "cpBackstayJMS".)

Figure 3 e*Way Connection



In the configuration file, the Connection Type should be set to "Topic", since the sample is using the Publish/Subscribe model.

Figure 4 Connection Type



The server name is the computer name on which the JMS Server (Oracle SeeBeyond JMS IQ Manager) is running. This is also the machine on which the Participating Host is installed. The Host name is the same as the server name.

4 Using the Components editor, create a new e*Way.

5 Display the new e*Way's properties.

6 On the **General** tab, under **Executable File**, click **Find** and select **stceway.exe**.

7 Click **OK** to close the properties dialog box, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 3**. The setup and requirements of schemas using the stceway.exe e*Way are discussed in **Chapter 4**.

8 Each e*Way has a Collaboration associated with it. Each Collaboration must have an associated source/destination.

Note: *Once the e*Way is installed and configured, it must incorporated into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before the e*Way can perform it's intended function. For more*

information about any of these procedures, please see the online Help system.

*For more information about configuring e*Ways or using the e*Way Editor, see the* ***e*Gate Integrator User's Guide****.*

3.1.1. Multi-Mode e*Way Configuration Parameters

e*Way configuration parameters are set using the e*Way Editor.

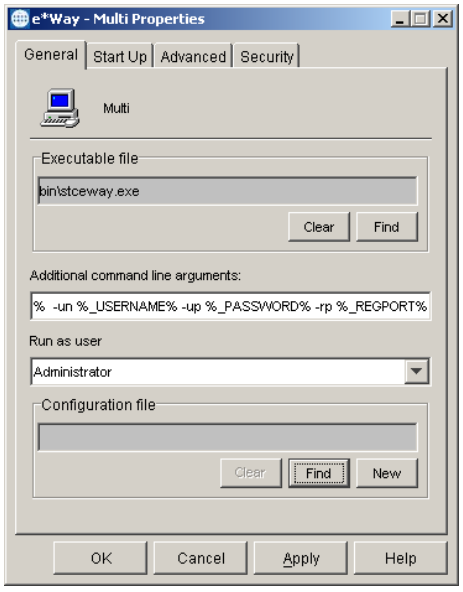
3.2 Multi-Mode e*Way Configuration

Multi-Mode e*Way properties are set using the Schema Designer.

To create and configure a New Multi-Mode e*Way:

1. Select the Navigator's Components tab.
2. Open the host and control broker on which you want to create the e*Way.
3. On the Palette, click on the **Create a New e*Way** button.
4. The New e*Way Component window opens. Enter the name of the new e*Way, then click **OK**.
5. Right-click the new e*Way and select **Properties** edit its properties.

Figure 5 Multi-Mode e*Way Properties



6. When the e*Way Properties window opens, click on the **Find** button beneath the **Executable File** field, and select an executable file. For the purposes of the sample select **stceway.exe** (**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 current configuration. Close the **.cfg** file and select **OK** to close the e*Way Properties Window.

Multi-Mode e*Way Configuration Parameters

The Multi-Mode e*Way configuration parameters are arranged in the following sections:

3.2.1. 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* is located on the Participating Host.

Required Values

A valid pathname.

Additional Information

The JNI dll name varies on different O/S platforms:

OS	Java 2 JNI DLL Name
Windows	jvm.dll
Solaris 2.6, 2.7, 2.8	libjvm.so

OS	Java 2 JNI DLL Name
HP-UX	libjvm.sl
AIX 4.3.3 and 5.1	libjvm.a

The value assigned may contain a reference to an environment variable. To do this, enclose the variable name within a pair of % symbols. For example:

```
%MY_JNIDLL%
```

Such variables are used when multiple Participating Hosts are used on different platforms.

Note: *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.*

CLASSPATH Prepend

Description

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

Required Values

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

Additional Information

If left unset, no paths are prepended to the CLASSPATH environment variable. Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

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

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

Required Values

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

Additional Information

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

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

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

Required Values

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

Initial Heap Size

Description

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

Required Values

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

Maximum Heap Size

Description

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

Required Values

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

Maximum Stack Size for Native Threads

Description

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

Required Values

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

Maximum Stack Size for JVM Threads

Description

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

Required Values

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

Disable JIT

Description

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

Required Values

YES or **NO**.

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

Remote Debugging port number

Description

Specifies the port number by which the e*Gate Java Debugger can connect with the JVM to allow remote debugging.

Required Values

An unused port number in the range 2000 through 65535. If not specified, the e*Gate Java Debugger is not able to connect to this e*Way.

Suspend option for debugging

Description

Allows you to specify that the e*Way should do no processing until an e*Gate Java Debugger has successfully connected to it.

Required Values

YES or **No**. YES suspends e*Way processing until a Debugger connects to it. NO enables e*Way processing immediately upon startup.

Auxiliary JVM Configuration File

Description

Specifies an auxiliary JVM configuration file for additional parameters.

Required Values

The location of the auxiliary JVM configuration file.

3.2.2. General Settings

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

- **Rollback Wait Interval** on page 22
- **Standard IQ FIFO** on page 22

Rollback Wait Interval

Description

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

Required Values

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

Standard IQ FIFO

Description

Specifies whether the highest priority messages from all STC_Standard IQs will be delivered in the first-in-first-out (FIFO) order.

Required Values

Select **YES** or **NO**. YES indicates that the e*Way will retrieve messages from all STC_Standard IQs in the first-in-first-out (FIFO) order. NO indicates that this feature is disabled. NO is the configured default.

3.3 Configuring the Web Server Components

Each Web server requires different configuration. Consult your Web server documentation for more information.

3.3.1. Configuring Apache Web Server

Configuring the Web server to use the CGI e*Way components on Apache Web server

The Web server should run the client executable, **stccgi.exe**, when a request is received. It must also set the dynamic-load library path in order for **stc_msapi.dll**, **stc_msclient.dll**, and **stc_mscommon.dll** to be loaded by **stccgi.exe**.

1. Modify the Web server configuration file to include the dynamically loaded library path (LD_LIBRARY_PATH, SHLIB_PATH, LIBPATH, or PATH) which contains the path of the e*Gate API Kit JMS client dll files, **stc_msclient.dll**, **stc_mscommon.dll** and **stc_msapi.dll**.

   For example, the Apache Web server on Solaris should appear as:

   ```
   setenv LD_LIBRARY_PATH "/usr/egate/client/bin"
   ```

2. Copy the **stccgi.exe** and **mscgi.properties** to the CGI bin directory and modify **mscgi.properties** to configure the CGI executable.
3. Change the permission on **stccgi.exe**, **stc_msapi.dll**, **stc_msclient.dll** and **stc_mscommon.dll**, enabling them to be read and executed by the Web server.

Note: *Consult the Web server documentation for more information.*

4. Copy the test*.html file to the Web server's doc root.

   For example, the Apache Service directory is:

   ```
   Apache\htdocs.
   ```

5. Access the test*.html file from a Web browser, and send a file to the CGI Web Server e*Way server. If successful, the file you sent to the server is displayed. The URL used to access the stccgi.exe is:

   ```
   http://hostname:port/cgi-bin/stccgi.exe
   ```

   A sample HTML form used to access stccgi.exe appears below:

   ```
   <HTML>

   <FORM ACTION="cgi-bin/stccgi.exe" METHOD="POST"
   ENCTYPE="multipart/form-data">
   Multipart test
   <P>
   <TABLE>
     <TR>
       <TD><LABEL for="fname">First name: </LABEL>
       <TD>  <INPUT type="text" name="firstname" id ="fname">
     <TR>
       <TD><LABEL for="lname">Last name: </LABEL>
       <TD><INPUT type="text" name="lastname" id="lname">
   </TABLE>
       <LABEL for="email">email: </LABEL>
   ```

```
                <INPUT type="text" name="email"><BR>
        <INPUT type="radio" name="sex" value="Male"> Male<BR>
        <INPUT type="radio" name="sex" value="Female"> Female<BR>
        <LABEL for="filename">What files are you sending? </LABEL>
                <INPUT type="file" name="filename"><BR>
        <INPUT type="submit" value="Send"> <INPUT type="reset">
        </P>
</FORM>

</HTML>
```

3.3.2. Configuring IIS Web Server

Configuring the Web server to use the CGI e*Way components on IIS Web server

The Web server should run the client executable, **stccgi.exe**, when a request is received. It must also set the dynamic-load library path in order for **stccgi.exe** to load **stc_msapi.dll, stc_mscommon.dll** and **stc_msclient.dll**.

1 Create the cgi-bin directory in the Inetpub directory.

If the default IIS server installation was used, the root directory is:

```
\inetpub
```

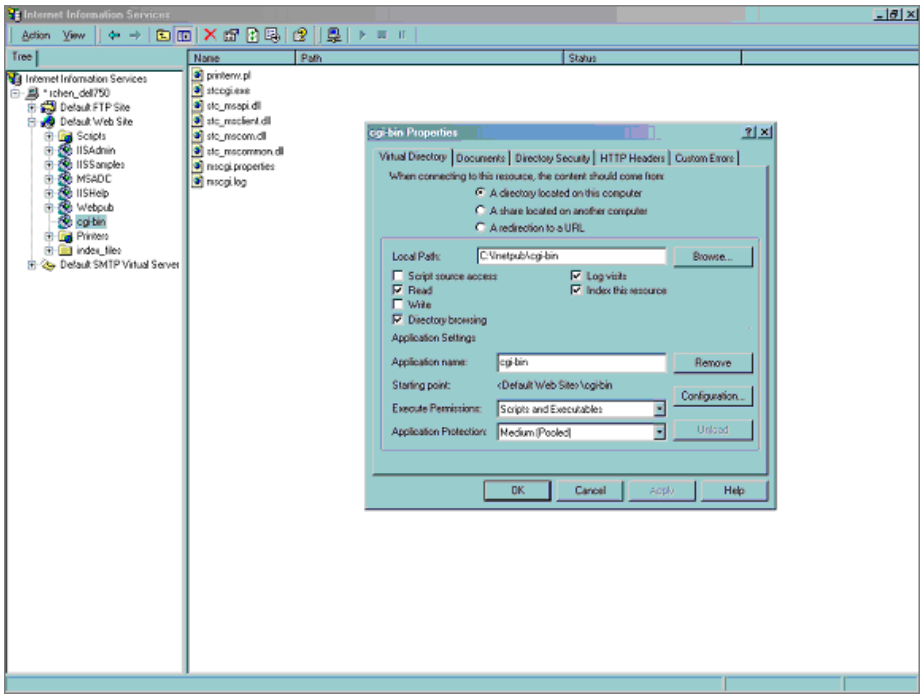
For example, create the following directory:

```
\inetpub\cgi-bin
```

Note: *The creation of this directory is not mandatory, but is recommended to assist with maintenance and to conform to common industry practices. The cgi-bin directory is used to store all cgi applications.*

2 Using the Internet Information Services Manager, go to Start\Settings\Control Panel\Administrative Tools\Internet Services Manager, or using Internet Information Services snap-in in Win 2K advanced server, create a virtual directory. To create a virtual directory, select Default Web Site in IIS manager, right click and choose action New\Virtual Directory. Alias: cgi-bin; Directory: C:\inetpub\cgi-bin, (Use the same directory as created in the preceding step) Access permissions: Read, Run Scripts and Execute.

Figure 6 IIS Internet Services Manager



3 Copy the **stccgi.exe**, **stc_msapi.dll, stc_msclient.dll** and **stc_mscommon.dll** to the CGI bin directory.

4 Create/copy a test.html file to:

```
C:\Inetpub\wwwroot
```

or the doc root that was configured for IIS server.

5 Modify **mscgi.properties** to configure the CGI executable. Change the permission on **stccgi.exe**, **stc_msapi.dll**, **stc_msclient.dll** and **stc_mscommon.dll**, enabling them to be read and executed by the Web server.

For IIS, ensure that for the directory created above (cgi-bin), the Execute Permissions setting is set to “Scripts and Executables”. To modify this setting, go to Internet Service Manager, click on the Web site (for example, Default Web Site), right-click on Scripts and select Properties. In the Scripts Properties window, click on the Virtual Directory tab. Select “Scripts and Executables” on the Execute Permissions scroll menu. Select OK, then restart the Web server.

Note: *Consult the Web server documentation for more information.*

6 Access the test*.html file from a Web browser and send a file to the CGI Web Server e*Way server. If successful, the file you sent to the server is displayed. The URL used to access the stccgi.exe is:

```
http://hostname/cgi-bin/stccgi.exe
```

A sample HTML form used to access stccgi.exe appears as follows:

```
<HTML>

<FORM ACTION="/cgi-bin/stccgi.exe" METHOD="POST"
```

```
ENCTYPE="multipart/form-data">
Multipart test
<P>
<TABLE>
  <TR>
    <TD><LABEL for="fname">First name: </LABEL>
    <TD>  <INPUT type="text" name="firstname" id ="fname">
  <TR>
    <TD><LABEL for="lname">Last name: </LABEL>
    <TD><INPUT type="text" name="lastname" id="lname">
</TABLE>
    <LABEL for="email">email: </LABEL>
              <INPUT type="text" name="email"><BR>
    <INPUT type="radio" name="sex" value="Male"> Male<BR>
    <INPUT type="radio" name="sex" value="Female"> Female<BR>
    <LABEL for="filename">What files are you sending? </LABEL>
              <INPUT type="file" name="filename"><BR>
    <INPUT type="submit" value="Send"> <INPUT type="reset">
    </P>
</FORM>

</HTML>
```

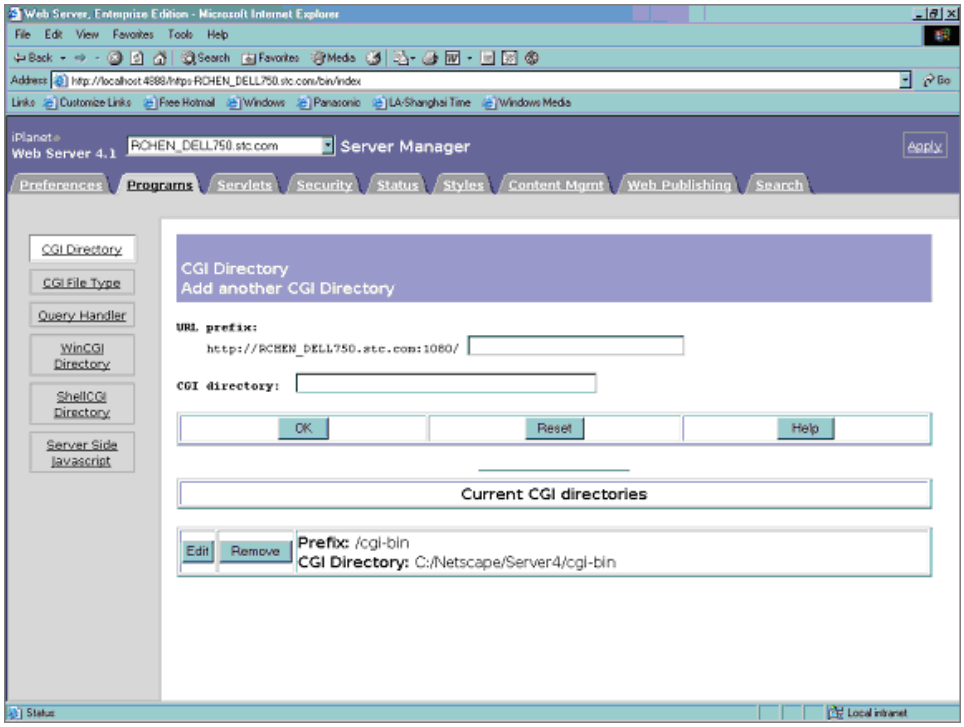
3.3.3. Configuring iPlanet Web Server

To configure the Web server to use the CGI e*Way Web server components on iPlanet Web server

The Web server should run the client executable, **stccgi.exe**, when a request is received. It must also set the dynamic-load library path in order for **stccgi.exe** to load **stc_msapi.dll, stc_mscommon.dll** and **stc_msclient.dll**.

1 Access the iPlanet Admin Web site. Select Server Manager and configure Programs\ CGI Directory as displayed in Figure 7:

Figure 7 iPlanet Server Manger



2 Copy the **stccgi.exe**, **stc_msapi.dll, stc_msclient.dll** and **stc_mscommon.dll** to the following CGI bin directory:

```
/NetScape/Server4/cgi-bin
```

3 Modify **mscgi.properties** to configure the CGI executable. Change the permission on **stccgi.exe**, **stc_msapi.dll**, **stc_msclient.dll** and **stc_mscommon.dll**, enabling them to be read and executed by the Web server.

Note: *Consult the Web server documentation for more information.*

4 Access the test*.html file from a Web browser and send a file to the CGI Web Server e*Way server. If successful, the file you sent to the server is displayed. The URL used to access the stccgi.exe is:

```
http://hostname/cgi-bin/stccgi.exe
```

A sample HTML form used to access stccgi.exe appears as follows:

```
<HTML>

<FORM ACTION="/cgi-bin/stccgi.exe" METHOD="POST"
ENCTYPE="multipart/form-data">
Multipart test
<P>
<TABLE>
  <TR>
    <TD><LABEL for="fname">First name: </LABEL>
    <TD>  <INPUT type="text" name="firstname" id ="fname">
  <TR>
    <TD><LABEL for="lname">Last name: </LABEL>
    <TD><INPUT type="text" name="lastname" id="lname">
```

```
</TABLE>
    <LABEL for="email">email: </LABEL>
              <INPUT type="text" name="email"><BR>
    <INPUT type="radio" name="sex" value="Male"> Male<BR>
    <INPUT type="radio" name="sex" value="Female"> Female<BR>
    <LABEL for="filename">What files are you sending? </LABEL>
              <INPUT type="file" name="filename"><BR>
    <INPUT type="submit" value="Send"> <INPUT type="reset">
    </P>
</FORM>

</HTML>
```

3.3.4. Modifying the mscgi.properties File

The **mscgi.properties** file must be edited before the CGI e*Way is run. The file contains information pertaining to the JMS Connection, CGI Data, and Logging values.

The properties file is loaded by the Oracle SeeBeyond JMS CGI. Each property is a name/value pairing. The name uniquely identifies the property. The value is the content associated with that name. The name is separated from the value with the ':' character.

Important: *DO NOT change names.*

JMS Connection Section

Host

The name of the host on which the Message Service is running. The Oracle SeeBeyond JMS IQ Manager acts as the Message Service (server). If Host is not specified, then localhost is the default value.

```
Host:localhost
```

Port

The port at which the Message Service is listening for connections. If the port is not specified, then 7555 is the default value.

```
Port:24053
```

RequestReply

Selects the JMS delivery mode as Request/Reply or Publish/Subscribe. Specify **True** for Request/Reply mode; See **"Timeout"** to configure reply Timeout. Specify **False** for Publish or Send mode.

```
RequestReply:True
```

Timeout

Timeout for Request/Reply. This specifies the timeout in milliseconds to wait for the reply. This is used only for the Request/Reply mode. See **"RequestReply"**.

```
Timeout:60000
```

TopicRequest

Selects the JMS mode as Topic or Queue request. Specify **True** for Topic requests. See **"Topic"** to configure the JMS Topic. Specify **False** for Queue requests. See Queue to configure the JMS Queue. The default is Topic request.

```
TopicRequest:True
```

Topic

The JMS Topic that the CGI uses to send a message to the Message Service. There is no default value for this property. This must be specified for Topic requests. See **"TopicRequest"**.

For example, Topic:etRequestReplyTopic121. This is the same value as the ETD type name, which the participating host receives. Refer to the sample schema for more information.

```
Topic:etwebRequestETDTopic
```

Queue

The JMS Queue that CGI uses to send a message to the Message Service. There is no default value for this property. This must be specified for Queue requests. See **"TopicRequest"**. For example, Queue:etRequestReplyQueue.

```
Queue:etwebRequestETDTopic
```

ClientID

The Client ID to use for the JMS connection. For example, ClientID:SeeBeyondMSCGI.

```
ClientID:SeeBeyondMSCGI1
```

CGI Data Section

If the **webRequest ETD** is used, the **EnvInBody** value must be **False** and **EnvAsProps** value must be **True**. For all other combinations a Custom ETD must be created. See **Figure 8 on page 30** for more information.

EnvInBody

Include the CGI Environments in the message body. If **True**, then each CGI environment is added before the CGI message body. Each environment is a name/value pair with '=' separating the name from the value. Each environment is separated by a new line. If **False**, then the CGI environments are not added to the message body. See **"EnvEnd"**.

EnvEnd

The text denoting the end of the Environment values. If EnvInBody is set to **True**, EnvStart is used to separate the message body from the environments. See **"EnvInBody"**. Do not change this value.

```
EnvEnd:<--End Environments-->
```

EnvsAsProps

IF **True**, Include the CGI Environment as JMS Properties. Each CGI environment is added to the JMS message as a JMS string property. If the webRequestETD is also used to receive the message the value of the JMS property is returned in the node

"environmentVariable". When using a Custom ETD, and not using webRequestETD, to receive the message, then call the readProperty method. For example:

```
getin().readProperty(CONTENT_TYPE)
```

Figure 8 CGI Data - EnvInBody/EnvAsProps

EnvInBody	EnvAsProps	Application
True	True	**Used with a CustomETD**. The JMS message is: all cgi env variables as name/value pair with name=value, followed by EnvEnd, followed by content body (if http post) or nothing (if http get). If the Custom ETD does not read these JMS properties, unused additional data is sent.
False	True	**webRequestETD can be used**, or a Custom ETD can be created for more options. The JMS message should appears as follows: <--End Environments--> followed by the content body (if http-post) or nothing (if http-get), followed by the tag <--End Environments--> regardless of what is defined in EnvEnd.
True	False	**Used with a Custom ETD**. The JMS message is the same as it is for True/True, except that it lacks all of the JMS properties. EnvEnd is used to separate the header from content body.
False	False	**Used with a Custom ETD**. The JMS message is: EndEnv followed by content body (if http post) or nothing (if http get).

ReadChunksize

When cgi does **read** from standard in, ReadChunksize specifies the chunk size, in bytes, of data to be read.

If you specify 1024 then cgi reads 1024 bytes of data once a time. If the content length is less than the chuck size, CGI just does read based on the content length. The default internal read chuck size is 409600 bytes. ChunkSize is an integer value, the maximum you can specify is 2147483647 bytes.

```
ReadChunksize:409600
```

WriteChunksize

When cgi does write to standard out, WriteChunksize specifies the chunk size in bytes, of the data to be written at one time. If you specify 1024 then cgi writes 1024 bytes of data once a time. The default internal write chuck size is 409600 bytes. ChunkSize is an integer value, the maximum you can specify is 2147483647 bytes.

```
WriteChunksize:409600
```

Log Section

LogFile

The log filename. Messages are logged into this file. See **"Trace"** to set the trace/log level.

```
LogFile:mscgi.log
```

Trace

The trace level to use for trace/debug. The following are valid values:

0 - Information
1 - Warning
2 - Error
3 - Fatal

```
Trace:0
```

3.3.5. Hex Dump vs. Text Dump

The CGI e*Way Web server componet provides "dumps" (the contents of each message written to the log file), of every inbound (request) and outbound (reply) message that it handles, provided that the Trace level in the mscgi.properties file is set to "0".

There are two types of dumps that can occur, a text dump and a hex dump. Text dumps are formatted into standard text. Hex dumps are formatted into lines of 16-bytes with two representations each, in it's own section. The first section is the hex representation of 16-bytes, followed by the second, which contains the ASCII representation of the same 16-bytes. If any byte is non-printable, a dot is substituted. The type of dump which occurs, is determined by the content-type of the message.

If the inbound message to the CGI e*Way (the date read from the CGI stdin) is any content-type other than text/*, a hex dump occurs. If the content-type is text/*, a text dump occurs. The inbound hex dump does not include any environment variables from HTTP server, such as CONTENT_TYPE, CONTENT_LENGTH, PATH or HTTP_ACCEPT.

If the **Default Outgoing Message Type** parameter in the e*Way Connection configuration is set to publish "bytes" message, a hex dump occurs. If it is set to publish "text" messages, a text dump occurs.

Sample Hex Dump

```
MS-CGI (main.cxx:755) I:  2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D  ----------------
MS-CGI (main.cxx:755) I:  2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 37 64 32  -------------7d2
MS-CGI (main.cxx:755) I:  31 32 35 37 35 30 33 38 36 0D 0A 43 6F 6E 74 65  125750386..Conte
MS-CGI (main.cxx:755) I:  6E 74 2D 44 69 73 70 6F 73 69 74 69 6F 6E 3A 20  nt-Disposition:
MS-CGI (main.cxx:755) I:  66 6F 72 6D 2D 64 61 74 61 3B 20 6E 61 6D 65 3D  form-data; name=
MS-CGI (main.cxx:755) I:  22 66 69 72 73 74 6E 61 6D 65 22 0D 0A 0D 0A 66  "firstname"....f
MS-CGI (main.cxx:755) I:  69 72 73 74 6E 61 6D 65 0D 0A 2D 2D 2D 2D 2D 2D  irstname..------
```

Sample Text Dump

If you submit http request with content type text/*, you get text dump.

```
MS-CGI (main.cxx:701) I: SUCCESSFULLY Set CGI STDIN to BINARY MODE
MS-CGI (main.cxx:720) I: Read accumulated data size [8192] and actual read size [8192]
MS-CGI (main.cxx:731) I: Dump data chunk :
MS-CGI (main.cxx:732) I:
```

Text dump (by setting e*Way Connection to publish text message)

```
MS-CGI (Reply.cxx:167) I: Returning message of type TextMessage
MS-CGI (main.cxx:273) I:  ~!@#$%^&*()_+`-={}|[]\:";'<>?,./
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIGKLMNOPQRSTUVWXYZ
```

Chapter 4

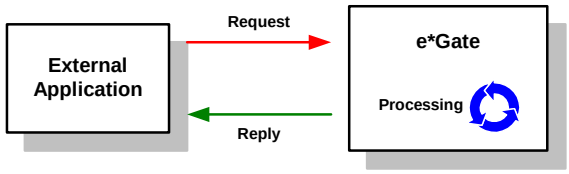
Implementation

4.1 The Request/Reply Model

All the applications of the CGI Web Server e*Way are based upon the "Request/Reply" concept. At a high-level, this works as follows:

1. Request/Reply, where data is sent to the e*Gate system and a response is returned.
2. Send-only or Fire and Forget, where data is sent to e*Gate but no data is returned. (See Figure 9.)

Figure 9 The Request/Reply concept

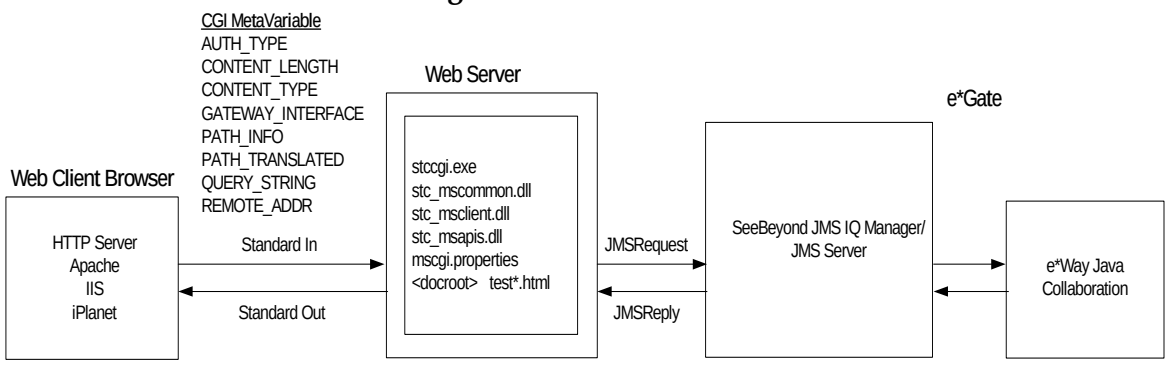


4.1.1. Request/Reply and the CGI Web Server e*Way Participating Host Components

The CGI Web Server e*Way Participating Host component is a Multi-Mode e*Way that uses a proprietary IP-based protocol to multi-thread Event exchange between the e*Way and external systems or other e*Gate components.

Figure 10 illustrates how the e*Way receives data from an external application and returns processed data to the same application. (See Figure 10.)

Figure 10 Data flow



1 The user accesses one of the test*.html files from a Web browser, sending a file to the CGI Web Server e*Way server, using either HTTP Post or Get.

2 A JMSRequest is made, the JMSReplyTo property is assigned, and a TemporaryTopic subscriber is created.

3 The JMSRequest is passed to the JMS Server (also known as the Oracle SeeBeyond JMS IQ Manager), which then uses the etwebRequestETDTopic Event Type name to forward the text message to an e*Way Collaboration.

4 Collaborations within the e*Way perform any appropriate processing that may be required, and route the processed Events to other destinations (such as to an external system for additional data retrieval or processing and then back to the Web server as a Reply, using TemporaryTopic).

5 The Web server gets the content from TemporaryTopic, and replies to the Web client.

4.1.2. The Request/Reply Sample

The sample schema can be found in the e*Gate installation CD-ROM in the **samples/ewcgi/Java** directory.

e*Gate Request/Reply Sample Set up

Request Reply Sample

1 Install the CGI Web Server e*Way Server add on.

2 Import the sample schema.

3 These instructions also appear in the "readme" file in the CGI Web Server e*Way samples directory (**samples/ewcgi/java**).

Once the client and server are set up, you can test the entire system using a Web browser:

1 Start the Control Broker. This also starts the JMS Server (Oracle SeeBeyond JMS IQ Manager).

2 Start the desired e*Way. In the sample, each e*Way demonstrates different functionality.

3 Use a browser to open the **test*.html** file.

4 Fill out and submit the form.

5 Confirm that the **stccgi.exe** returned the form data as expected. You can view the mscgi.log file , located in cgi-bin, to see the content sent out/received by stccgi.exe from the JMS Server.

Sample Functionality

Separate e*Ways handle various activities. There are four basic steps involved in a standard Request/Reply schema:

1 Define the content.

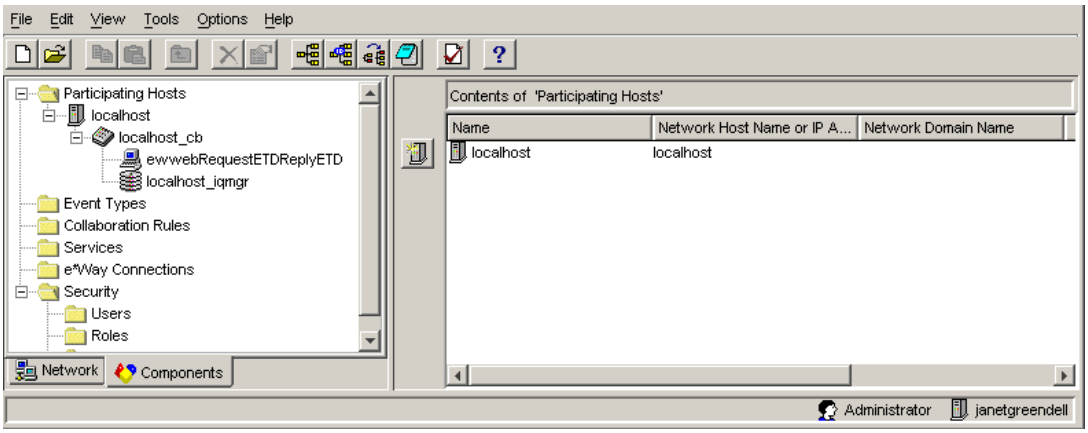
2 Populate the webReplyETD payload.

3 Call the send method in the webReplyETD.

4 Using the JMSReplyTo property, send the input to the webRequestETD.

Once successfully installed, the Schema Designer opens to the following:

Figure 11 webETD_CGI Schema Designer Components View



The sample is comprised of one e*Way:

- ewwebRequestETDReplyETD

ewwebRequestETDReplyETD

The ewwebRequestETDReplyETD e*Way receives a block of data, parses the input into webRequestETD, allows the creation of webReplyETD, then marshals the data into a block, and sends the block of data back.

crRequestReply_webRequestETDReplyETD

The Collaboration Rule associated with the ewwebRequestETDReplyETD e*Way performs the following:

1 Provides the HTTP content type for the reply message.

2 Gets the specified HTTP headers that CGI passes back to the Web server.

3 Demonstrates how to get the decoded inbound payload.

4 Demonstrates how to set outbound encoding. (Encoding is performed automatically and is not required.)

5 Shows charSet encoding/decoding activity. You perform a get of the decoded charSet payload from textStringPayLoad. If you require outbound encoding of the charSet, set the outbound charSet attributes.

6 Demonstrates how to get multiPart payload, and how to drill down on nested multiPart payload.

7 Demonstrates how to build an outbound multiPart payload.

8 Demonstrates how to get HTTP_GET URLencoded name value pairs. The URLencoded string is decoded into name value pairs.

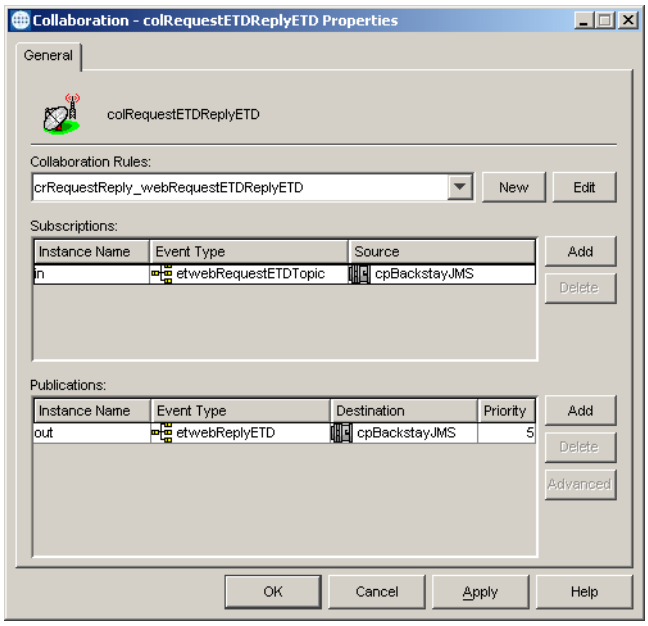
9 Sends the HTTP-enabled data to the specified reply topic.

colRequestETDReplyETD

The Collaboration that ties the e*Way and the Collaboration Rule together is "colRequestETDReplyETD". The Collaboration must have an Event Type and Source/Destination defined. For colRequestReply, the Collaboration subscribes to etWebRequestETD as a temporary topic. Keep in mind that stccgi.exe has been configured to publish etWebRequestETD topic. This is where the publication and subscription of the topic are seen. The temporary topic serves as a reply to the request, and is therefore transparent to the user. The exact temporary topic is stored in the input Event JMSReplyTo node.

The Collaborations subscribe and publish to the same external source/destination "cpBackstaryJMS" (the e*Way Connection).

Figure 12 colRequestETDReplyETD



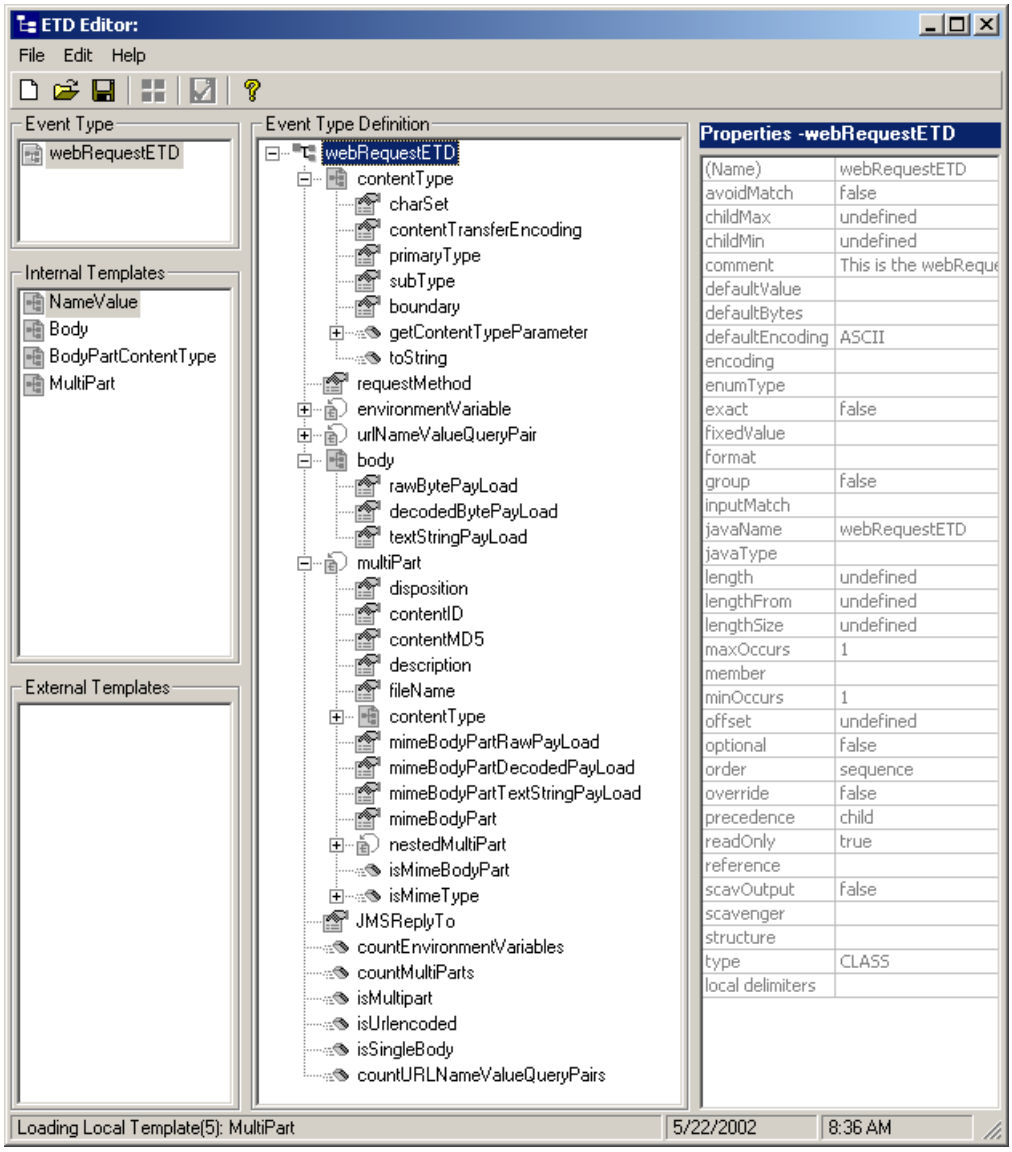
4.1.3. Event Type Definitions

As discussed in **Request/Reply and the CGI Web Server e*Way Participating Host Components** on page 33, the CGI Web Server e*Way maintains the "**JMSReplyTo**" property. "**JMSReplyTo**" must be set to facilitate the reply data flow through the e*Gate system.

webRequestETD

The **webRequestETD** Event Type Definition (ETD) which can be used within a Request/Reply schema is shown below. The **webRequestETD** is designed to be "read only," and the **webReplyETD** is designed to be "write only."

Figure 13 The webRequestETD



Node Descriptions

contentType

The top level node "**contentType**" represents the CONTENT_TYPE meta variable. If the message can be defined as a discrete-type media, such as **text/***, **image/***, the content type shows up at the top level.

The following subnodes appear below it.

charSet

If the charSet is not supported, or is misspelled, the decoding can not take place. In that case, use **rawBytePayLoad**, or **decodedBytePayLoad**. The charSet encoding table is available at:

http://www.ingrid.org/java/i18n/encoding/table.html

The value can be null if the message doesn't use the charset attribute.

contentTransferEncoding
Indicates whether the content is encoded (base64, binary, and so forth) the field is not case sensitive. It may be return a null string if not transfer encoded.

primaryType
Indicates the primary content type. For example, text in text/xml.

subType
Indicates the sub type of the content. For example, xml in text/xml.

boundary
If it is **multipart/*** data, this node carries the value of the boundary. If the message is nested multipart, this simply provides the top level boundary.

getContentTypeParameter
Provides a method to be called to get any other content type attributes, such as **ContentID**, may also return a null string, if the attribute is not defined.

requestMethod
Returns the HTTP method (for example, POST or GET).

environmentVariables

A top level node that contains an array of cgi variables received or passed to **stccgi** (for example HTTP_ACCEPT).

name
Return the name of the environment variable.

value
Returns the value of the environment variable.

urlNameValueQueryPair

A top level node that contains an array. Check the count of this array; if it returns a value equal to zero, the content is not a URL encoded value pair. If HTTP server received a get command, a URL encoded query string is received. The ETD decodes the string and populates the name value pair for use directly.

name
The name of the part. For example:

```
firstname
```

value
The value of the part. For example:

```
myfirstname
```

body

A top level node that contains the body of the content for discrete types, such as text/* or image/*.

The following subnodes appear below it.

rawBytePayLoad
Contains raw byte data.

decodedBytePayLoad

If the content indicates that it has been transfer encoded in the **contentType** header, the content-transfer-encoding attribute decodes the rawBytePayload. The result is the decoded byte array, while rawBytePayLoad still contains the original encoded byte data.

textStringPayLoad

If the content primary type indicates that it is text based, a string is created from rawBytePayLoad (or decodedBytePayLoad, if the content has been transfer encoded). The resulting text string is a Java internal representation of the original text string, in the designated charSet character encoding.

For example, a byte stream of data is received that is EUC_JP character encoded, the textStringPayLoad produces a Java internal string representation (Unicode) of that Japanese character stream. It is not in EUC_JP encoding, but in Java Unicode.

If the charSet is not recognized by the java decoder, (for example, you misspelled EUC_JP to ECU_JP) then you are not provided with a textStringPayLoad. You can access the data from rawBytePayLoad or decodedBytePayLoad.

If the original message contains a content transfer encoded text string, the rawBytePayLoad, decodedBytePayLoad, and textStringPayLoad all contain data.

If you specify the content type as application/xml, rather than text/xml, this media form is not recognized as a text type. textStringPayLoad is not populated, even though the content body is a text based byte array. In this case the content must be retrieved from rawBytePayLoad or decodedBytePayLoad (if content transfer encoded).

If the content type is not text/* and is not multipart/*, the payload must be retrieved via rawBytePayLoad or decodedBytePayLoad (if content transfer encoded).

For example, to use the Content-Type: text/plain;Content-Transfer-Encoding="Quoted-Printable" with the text "Now is the time for all folk to come to the aid of cgieway", the quoted-printable transfer encoding appears as below:

```
Now is the time=
For all folk to come=
To the aid of cgieway=
```

For more information, refer to RFC2045 at:

http://www.ietf.org/rfc/rfc2045.txt.

Per RFC2045:

```
Content-Type:text/plain; charset=EUC_JP
Content-transfer-encoding: base64
```

In accordance with RFC2045, the body is a base64 US_ASCI encoding of data that was originally in EUC_JP. After unmarshalling via webRequestETD, the textStringPayLoad provides a Java string Unicode encoding of the Japanese content originally in EUC_JP.

multiPart

A top level node used in conjunction with multipart/* content type data. The following subnodes are available:

disposition
The content-disposition of each body part in the multipart array. Check for null values. For example, multipart/form data has top level content type populated as multipart/form:boundary=____. Each part has disposition form data.

contentID
Contains the **ContentID** attributes. Check for null values.

contentMD5
Returns the value of the ”content-MD5” header field. Returns null if this field is unavailable or absent. MD5 is a 128 bit digital finger print.

description
Returns the “content-Description” field.

filename
Returns the value if the content contains a filename, otherwise returns null.

contentType
Allows access to all contentType parameters. Similar to the top level contentType node, contentType indicates the break down of attributes for this body part. Check for null values.

mimeBodyPartRawPayLoad
Contains the raw byte data.

mimeBodyPartDecodedPayLoad
If the content indicates that it has been transfer encoded in the contentType header, content-transfer-encoding attribute decodes the rawBytePayLoad, returning a decoded byte array. In this case, the mimeBodyPartRawPayLoad still contains the encoded byte data.

mimeBodyPartTextStringPayLoad
If the content primary type indicates that it is text data, a string is created from the mimeBodyPartRawPayLoad (or the mimeBodyPartDecodedPayLoad if the content is transfer encoded). The resulting text string is a Java internal representation of the original text string, in the designated charSet character encoding.

For example, a byte stream of data is received that is EUC_JP character encoded, the textStringPayLoad produces a Java internal string representation (Unicode) of that Japanese character stream. It is not in EUC_JP encoding, but in Java Unicode.

If the original message contains a content transfer encoded text string, the mimeBodyPartRawBytePayLoad, mimeBodyPartDecodedBytePayLoad, and mimeBodyPartTextStringPayLoad all contain data.

If the content type is specified as application/xml, rather than text/xml, this media form is not recognized as a text type. **mimeBodyPartTextStringPayLoad** is not populated, even though the content body is a text based byte array. In this case the content must be retrieved from **mimeBodyPartRawBytePayLoad** or **mimeBodyPartDecodedBytePayLoad** (if content transfer encoded).

If the content type is not text/* and is not multipart/*, the payload must be retrieved via mimeBodyPartRawBytePayLoad or mimeBodyPartDecodedBytePayLoad (if content transfer encoded).

For example, to use the Content-Type: text/plain;Content-Transfer-Encoding="Quoted-Printable" with the text "Now is the time for all folk to come to the aid of cgieway", the quoted-printable transfer encoding appears as below:

```
Now is the time=
For all folk to come=
To the aid of cgieway=
```

For more information, refer to RFC2045, (page 16) at **http://www.ietf.org/rfc/rfc2045.txt**.

Per RFC2045:

```
Content-Type:text/plain; charset=EUC_JP
Content-transfer-encoding: base64
```

In accordance with RFC2045, the body is a base64 US_ASCI encoding of data that was originally, in EUC_JP. After unmarshalling via webRequestETD, the textStringPayLoad provides a Java string Unicode encoding of the Japanese content originally in EUC_JP.

mimeBodyPart
Returns an object of javax.Mail.Internet.MimeBodyPart. This value can be accessed via multipart nesting. This provides the means to access a raw object handle.

nestedMultiPart
If the multiPart contains another multipart, as an associated body part, this node allows for drilling down to the nested multiPart. If it is not nested, it returns null. The **mimeBodyPartRawPayLoad** and **decodedPayLoad** are maintained at the same time.

isMimeBodyPart
Queries whether the body part is of a mime type.

isMimeType
Queries whether the body part is of a certain mime type.

JMSReplyTo

Gets the **JMSReplyTo property**.

countEnvironmentVariable

Returns an integer, indicating the number of environment variables.

countMultiparts

Returns an integer, indicating the number of body parts.

isMulitpart

Returns true or false, indicating whether multipart or not.

isUrlencoded

Returns true or false, indicating whether the data is URL encoded.

isSingleBody

Returns true or false, indicating whether the data consists of a single body part.

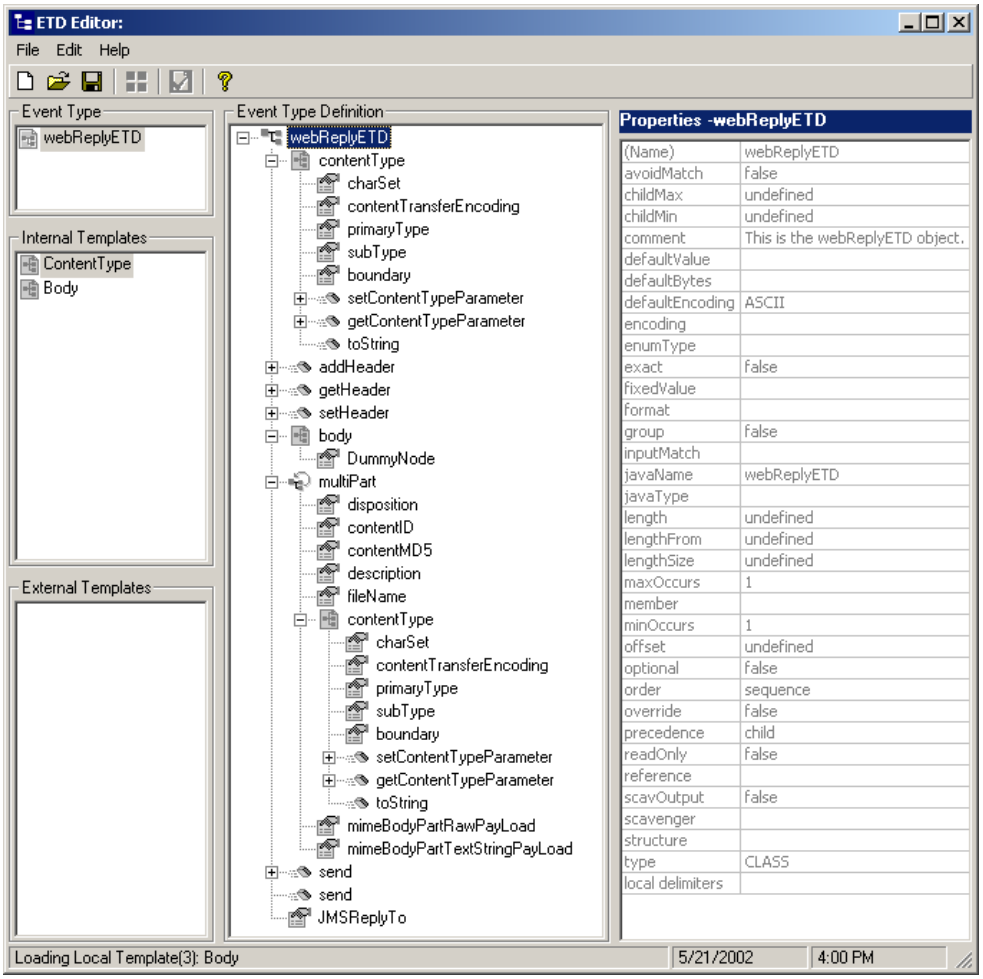
countURLNameValueQueryPairs

Returns an integer, indicating the number of name/value pairs.

webReplyETD

The **webReplyETD** Event Type Definition (ETD) is used within a Request/Reply schema. The **webReplyETD** mirrors the **webRequestETD**. The **webReplyETD** is "write" only, while the **webRequestETD** is "read" only. There are functions availabe to allow you to read the node value, but the node value must be set before reading.

Figure 14 The webReplyETD



Node Description

contentType

The **CONTENT_TYPE** header that is returned to the HTTP client. This **contentType** is prepended to the body with two carriage return line feeds (CRLFs).

The following subnodes appear below it.

charSet
Setting this value brings about a byte array of text data as the output. The body node's **textStringPayLoad** must also be set.

contentTransferEncoding
Setting this value defines the content as encoded (such as base64). If the data is multipart, do not set transfer encoding at the top level.

primaryType
Sets the primary content type. For example, text in text/xml.

subType
Sets the sub type of the content. For example, xml in text/xml.

boundary
This value is only set if the user does not want to use the system defined boundary. Do not set the boundary if the data is not multipart data.

setContentTypeParameter
Sets a name/value pair in **contentType**.

getContentTypeParameter
Gets a name/value pair from **contentType**.

addHeader

A method that allows the user to add other http/cgi response headers, taking a name/value pair as the parameters. It is not necessary to add **content_type** headers here since that is already accomplished by the setting the **contentType** node. The ETD bundles the **contentType** headers behind the scenes.

getHeader

A method that allows the user to retrieve the header. It takes a name/delim pair as the parameters.

setHeader

A method that allows the user to set the header value. It takes a name/value pair as the parameters.

body

A top level node. If the data is of the discrete type, such as text/* or image/*, the body of content appears here. Set **rawBytePayLoad** or **textStringPayLoad** (one, not both).

The following subnodes appear under **body**:

rawBytePayLoad
If the data is not text based, the raw byte data is contained here.

textStringPayLoad
Set this payload for text string based data. If this value is set, the content type is not verified. The text string can be any language, as specified by :

http://www.ingrid.org/java/i18n/encoding/table.html

Specify the output charSet, if the string contains any non-default text, such as EUC_JP, SJIS, GB2312, ISO-8859-1. The desired output character should be specified, even when a conversion is not performed. For example, if the

textStringPayLoad contains java Unicode string EUC_JP, specify charSet=EUC_JP. The Unicode output results in a byte array that contains EUC_JP characters. If you set charSet=SJIS, the output results in Java Unicode encoding of the original EUC_JP string to a byte array that contains an SJIS character set.

multiPart

This is an array of body part. The ETD allocates a new body part the first time the body part attribute is accessed. Set each body part payload. If the boundary is set in the contentType node, that boundary value is used. If only header is set and not the payLoad, a marshalling exception occurs.

The following subnodes appear below:

disposition
Sets the disposition.

contentID
Sets the contentID.

contentMD5
Sets the MD5 signature.

description
Sets the description.

filename
Set the filename.

contentType
Allows the user to set the values of the following subnodes:

charSet
Setting this value produces a byte array of text data as output. The body node's **textStringPayLoad** must also be set.

contentTransferEncoding
Setting this value, cause a byte array of text data to result as the output. You must also then set the body node's **textStringPayLoad**.

primaryType
Sets the primary content type. For example, text in text/xml.

subType
Set the sub type of the content. For example, xml in text/xml.

boundary
This value is only set if you do not want to use the system defined boundary. Do not set if the data is not multipart.

setContentTypeParameter
Sets a name/value pair in contentType.

getContentTypeParameter
Gets a name/value pair from contentType.

mimeBodyPartRawPayLoad
Sets the body part payload to "raw".

mimeBodyPartDecodedPayLoad
Sets the body part payload to "decoded".

send

Sends the reply to the **JMSReply** property, taking the **replyTo** parameter string.

send

Sends the reply to the **JMSReply** property.

JMSReplyTo

Sets the **JMSReplyTo** property.

4.1.4. Collaboration Rules and the CGI Web Server Header

The Collaboration Rule provided in the sample (shown in Figure 15) demonstrates a number of possible behaviors.

1. Provide the HTTP content type for the reply/outbound message:

```
getout().getContentType().setPrimaryType( "text" )
getout().getContentType().setSubType("plain")
```

2. Get request/inbound CGI meta variables:

```
getin().getEnvironmentVariables(i).getName()
getin().getEnvironmentVariables(i).getValue()
```

3. Add another reply/outbound HTTP header:

```
getout().addHeader(getin().getEnvironmentVariables(i).getName(),ge
tin().getEnvironmentVariables(i).getValue())
```

4. Test/Get decoded inbound payload:

   Multipart test:

```
getin().getMultiParts(j).getContentType().getContentTransferEncodi
ng()==null
```

   Single body test:

```
getin().getContentType().getContentTransferEncoding()==null
```

   Multipart get decoded payload:

```
byte[]temppayload=getin().getMultiParts(outmlptindex).getMimeBodyP
artDecodedPayLoad()
```

   Single body decoded payload:

```
byte[]bytesinglebody=getin().getBody().getTransferCodePayLoad()
```

   If it is text message, it is always decoded for you with:

```
String strsinglebody=getin().getBody().getTextStringPayLoad()
```

5. Set charSet or content transfer encoding for reply/outbound and set the proper payload (the encoding is performed automatically):

```
getout().getContentType().setCharSet("SJIS")
getout().getMultiParts(outmlptindex).getContentType().setContentTr
ansferEncoding("base64")
getout().getMultiParts(outmlptindex).setMimebodyPartTextStringPayL
oad(getin().getMultiParts(j).getMimeBodyPartTextStringPayLoad())
```

```
getout().getMultiParts(outmlptindex).setMimeBodyPartRawPayLoad(tem
ppayload)
```

6 Get URL encoded name/value pair:

```
getin().isUrlencoded()
strbuf4.append(" url name:
"+getin().getURLNameValueQueryPairs(k).getName()+" value:
"+getin().getURLNameValueQueryPairs(k).getValue())
```

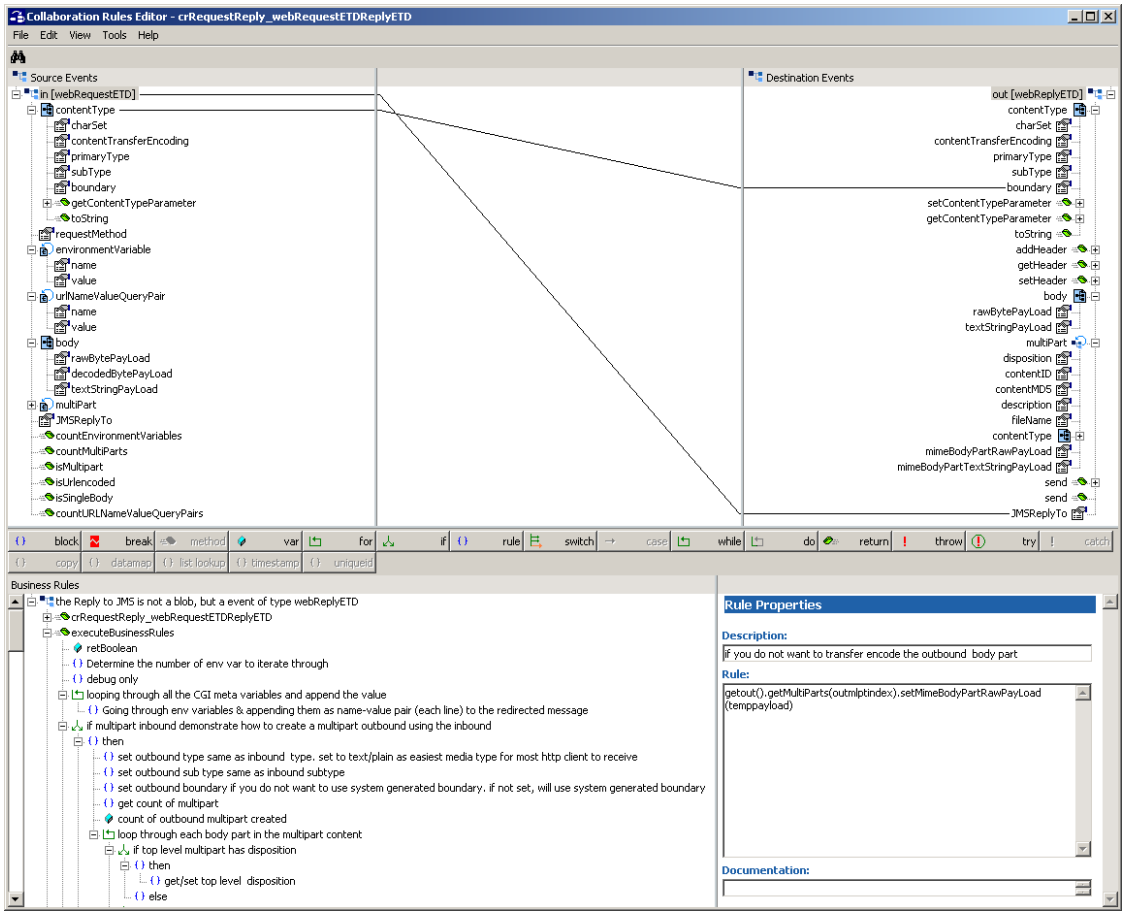
7 Send the HTTP-enabled data to the reply topic:

Manual publish:

```
getout().send(getout().getJMSReplyTo())
```

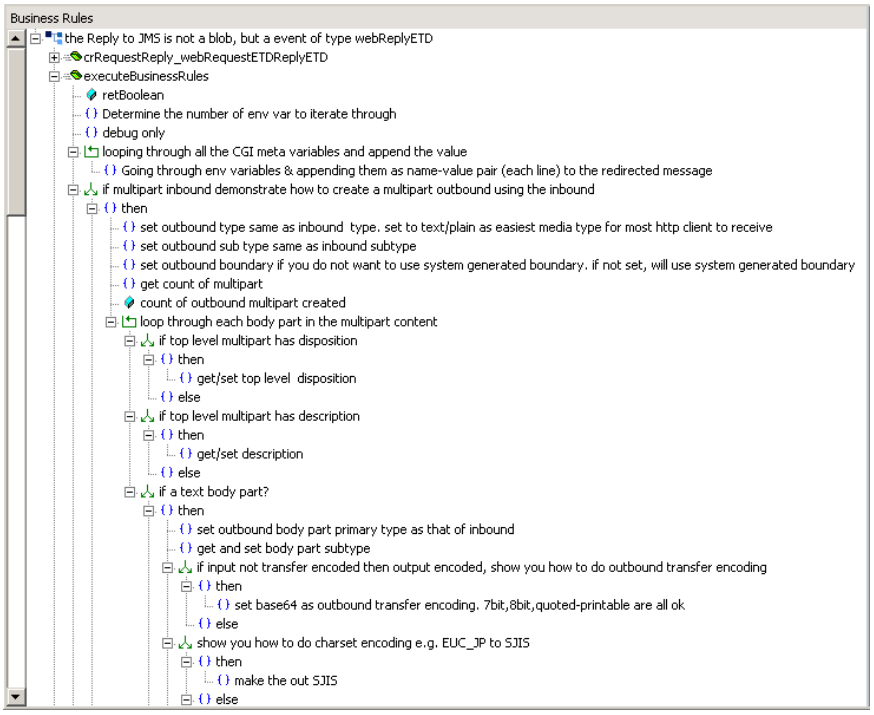
crRequestReply_webRequestETDReplyETD

Figure 15 Copying the Request/Reply field



A portion of the Business Rules that were implemented for crRequestReply_webRequestETDReplyETD Collaboration Rules sample, appear below:

Figure 16 Business Rules



The code itself can be viewed from the GUI. From the **View** Menu, select **View Java Code**. The code appears as follows:

```
import com.stc.common.collabService.*;
import com.stc.jcsre.*;
import com.stc.eways.util.*;
import java.io.*;
import java.sql.*;
import java.util.*;
import com.stc.eways.webETD.*;


   class crRequestReply_webRequestETDReplyETDBase extends JCollaboration
   {
      public crRequestReply_webRequestETDReplyETDBase()
      {
         super();
      }

      com.stc.eways.webETD.webRequestETD in = null;

      public com.stc.eways.webETD.webRequestETD getin()
      {
         return this.in;
      }

      com.stc.eways.webETD.webReplyETD out = null;

      public com.stc.eways.webETD.webReplyETD getout()
      {
         return this.out;
      }

      public void resetData() throws CollabConnException, CollabDataException
      {
         this.in = (com.stc.eways.webETD.webRequestETD) this.reset((ETD)this.getin());
         this.out = (com.stc.eways.webETD.webReplyETD) this.reset((ETD)this.getout());
      }
      public void createInstances() throws CollabConnException
      {
         this.in = (com.stc.eways.webETD.webRequestETD)
this.newInstance("com.stc.eways.webETD.webRequestETD", "in", ETD.IN_MODE);
         this.out = (com.stc.eways.webETD.webReplyETD)
this.newInstance("com.stc.eways.webETD.webReplyETD", "out", ETD.OUT_MODE);
      }
   }
```

```
   public class crRequestReply_webRequestETDReplyETD extends
crRequestReply_webRequestETDReplyETDBase implements JCollaboratorExt
   {
      public  crRequestReply_webRequestETDReplyETD()
      {
         super();
      }

      public boolean executeBusinessRules() throws Exception
      {
         boolean retBoolean = true;
         int EnvVarCnt = getin().countEnvironmentVariables();
         System.err.println("boundary:"+getin().getContentType().getMultipartBoundary());
         for( int i = 0;i < EnvVarCnt;i++)
         {

getout().addHeader(getin().getEnvironmentVariables(i).getName(),getin().getEnvironmentVariables(i)
.getValue());
         }
         if (getin().isMultipart())
         {
            getout().getContentType().setPrimaryType( "text"  );
            getout().getContentType().setSubType("plain");
            //getout().getContentType().setMultipartBoundary("xi-li-hu-tu-yi-xian-tian-kai-zi-bian-
zi-zao-boundary");
            int partscount = getin().countMultiParts();
            int outmlptindex = 0;
            for( int j = 0;j < partscount;j++)
            {
               if (getin().getMultiParts(j).getDisposition() != null)
               {

getout().getMultiParts(outmlptindex).setDisposition(getin().getMultiParts(j).getDisposition());
               }
               else
               {
               }
               if (getin().getMultiParts(j).getDescription() != null)
               {
                  getout().getMultiParts(outmlptindex).setDescription(
getin().getMultiParts(j).getDescription()     );
               }
               else
               {
               }
               if (getin().getMultiParts(j).getContentTypeString().indexOf("text") >= 0)
               {

getout().getMultiParts(outmlptindex).getContentType().setPrimaryType(getin().getMultiParts(j).getC
ontentType().getPrimaryType() );
                  getout().getMultiParts(outmlptindex).getContentType().setSubType(
getin().getMultiParts(j).getContentType().getSubType()  );
                  if (getin().getMultiParts(j).getContentType().getContentTransferEncoding() ==
null)
                  {

getout().getMultiParts(outmlptindex).getContentType().setContentTransferEncoding("base64");
                  }
                  else
                  {
                  }
                  if (getin().getMultiParts(j).getContentType().getCharSet() != null &&

getin().getMultiParts(j).getContentType().getCharSet().compareToIgnoreCase("EUC_JP") == 0)
                  {
                     getout().getMultiParts(outmlptindex).getContentType().setCharSet("SJIS");
                  }
                  else
                  {
                     if (getin().getMultiParts(j).getContentType().getCharSet() != null)
                     {
                        getout().getMultiParts(outmlptindex).getContentType().setCharSet(
getin().getMultiParts(j).getContentType().getCharSet()            );
                     }
                     else
                     {
                     }
                  }
                  getout().getMultiParts(outmlptindex).setDisposition(
getin().getMultiParts(j).getDisposition() );
                  getout().getMultiParts(outmlptindex).setDescription("add a sample description in
collab");
                  getout().getMultiParts(outmlptindex).setFileName(
getin().getMultiParts(j).getFileName()  );

getout().getMultiParts(outmlptindex).setMimeBodyPartTextStringPayLoad(getin().getMultiParts(j).get
MimeBodyPartTextStringPayLoad());
                  outmlptindex++;
               }
               else
               {
```

```
                    if (getin().getMultiParts(j).countNestedMultiPart() == 0)
                    {
                       if (getin().getMultiParts(j).getMimeBodyPartDecodedPayLoad() != null)
                       {
                          byte[] temppayload =
getin().getMultiParts(outmlptindex).getMimeBodyPartDecodedPayLoad();
                           getout().getMultiParts(outmlptindex).setMimeBodyPartRawPayLoad(temppayload);
                          getout().getMultiParts(outmlptindex).setDisposition(
getin().getMultiParts(j).getDisposition()    );
                          getout().getMultiParts(outmlptindex).setDescription(
getin().getMultiParts(j).getDescription()  );
                          getout().getMultiParts(outmlptindex).setFileName(
getin().getMultiParts(j).getFileName() );
                          getout().getMultiParts(outmlptindex).getContentType().setPrimaryType(
getin().getMultiParts(j).getContentType().getPrimaryType());
                          getout().getMultiParts(outmlptindex).getContentType().setSubType(
getin().getMultiParts(j).getContentType().getSubType()          );
                       }
                       else
                       {
                          byte[] temppayload = getin().getMultiParts(j).getMimeBodyPartRawPayLoad();
                         getout().getMultiParts(outmlptindex).setMimeBodyPartRawPayLoad(temppayload);
                          getout().getMultiParts(outmlptindex).setDisposition(
getin().getMultiParts(j).getDisposition()    );
                          getout().getMultiParts(outmlptindex).setDescription(
getin().getMultiParts(j).getDescription()  );
                          getout().getMultiParts(outmlptindex).setFileName(
getin().getMultiParts(j).getFileName() );
                          getout().getMultiParts(outmlptindex).getContentType().setPrimaryType(
getin().getMultiParts(j).getContentType().getPrimaryType());
                          getout().getMultiParts(outmlptindex).getContentType().setSubType(
getin().getMultiParts(j).getContentType().getSubType()          );
                       }
                       outmlptindex++;
                    }
                    else
                    {
                       int cntnestedpart = getin().getMultiParts(j).countNestedMultiPart();
                       for( int l = 0;l < cntnestedpart;l++)
                       {
                          if
(getin().getMultiParts(j).getNestedMultiPart(l).getMimeBodyPartDecodedPayLoad() != null)
                          {
                             byte[] temppayload =
getin().getMultiParts(j).getNestedMultiPart(l).getMimeBodyPartDecodedPayLoad();

getout().getMultiParts(outmlptindex).setMimeBodyPartRawPayLoad(temppayload);
                             getout().getMultiParts(outmlptindex).setDisposition(
getin().getMultiParts(j).getNestedMultiPart(l).getDisposition()    );
                             getout().getMultiParts(outmlptindex).setDescription(
getin().getMultiParts(j).getNestedMultiPart(l).getDescription()  );
                             getout().getMultiParts(outmlptindex).setFileName(
getin().getMultiParts(j).getNestedMultiPart(l).getFileName() );
                             getout().getMultiParts(outmlptindex).getContentType().setPrimaryType(
getin().getMultiParts(j).getNestedMultiPart(l).getContentType().getPrimaryType());
                             getout().getMultiParts(outmlptindex).getContentType().setSubType(
getin().getMultiParts(j).getNestedMultiPart(l).getContentType().getSubType()          );
                          }
                          else
                          {
                             byte[] temppayload =
getin().getMultiParts(j).getNestedMultiPart(l).getMimeBodyPartRawPayLoad();

getout().getMultiParts(outmlptindex).setMimeBodyPartRawPayLoad(temppayload);
                             getout().getMultiParts(outmlptindex).setDisposition(
getin().getMultiParts(j).getNestedMultiPart(l).getDisposition()    );
                             getout().getMultiParts(outmlptindex).setDescription(
getin().getMultiParts(j).getNestedMultiPart(l).getDescription()  );
                             getout().getMultiParts(outmlptindex).setFileName(
getin().getMultiParts(j).getNestedMultiPart(l).getFileName() );
                             getout().getMultiParts(outmlptindex).getContentType().setPrimaryType(
getin().getMultiParts(j).getNestedMultiPart(l).getContentType().getPrimaryType());
                             getout().getMultiParts(outmlptindex).getContentType().setSubType(
getin().getMultiParts(j).getNestedMultiPart(l).getContentType().getSubType()          );
                          }
                          outmlptindex++;
                       }
                    }
                 }
              }
           }
           else
           {
              System.err.println("Not Multiupart");
           }
           String strsinglebody;
           byte[] bytesinglebody;
           if (getin().isSingleBody())
           {
              getout().getContentType().setPrimaryType(getin().getContentType().getPrimaryType());
              getout().getContentType().setSubType(getin().getContentType().getSubType());
              if (getin().getBody().getTextStringPayLoad() != null)
```

```
                {
                   strsinglebody=getin().getBody().getTextStringPayLoad();
                   getout().getBody().setTextStringPayLoad(strsinglebody);
                   if (getin().getContentType().getCharSet() != null &&
                   getin().getContentType().getCharSet() .compareToIgnoreCase("EUC_JP")==0)
                   {
                      getout().getContentType().setCharSet("SJIS");
                   }
                   else
                   {
                      if (getin().getContentType().getCharSet() != null)
                      {
                         getout().getContentType().setCharSet(getin().getContentType().getCharSet());
                      }
                      else
                      {
                      }
                   }
                }
                else
                {
                   if (getin().getBody().getTransferCodePayLoad() != null)
                   {
                      bytesinglebody = getin().getBody().getTransferCodePayLoad();
                      getout().getBody().setRawPayLoad(bytesinglebody);
                   }
                   else
                   {
                      bytesinglebody = getin().getBody().getRawPayLoad();
                      getout().getBody().setRawPayLoad(bytesinglebody);
                   }
                   System.err.println("Single body does not have text payload");
                }
             }
             else
             {
                System.err.println("is not single body type");
             }
             StringBuffer strbuf4 = new StringBuffer();
             if (getin().isUrlencoded())
             {
                for( int k = 0;k  < getin().countURLNameValueQueryPairs();k++)
                {
                  strbuf4.append(" url name: "+getin().getURLNameValueQueryPairs(k).getName()+" value:
"+getin().getURLNameValueQueryPairs(k).getValue());
                }
                getout().getContentType().setPrimaryType("text");
                getout().getContentType().setSubType("plain");
                getout().getBody().setTextStringPayLoad(strbuf4.toString());
             }
             else
             {
                System.err.println("not url encoded request");
             }
             getout().setJMSReplyTo(getin().getJMSReplyTo());
             getout().send(getout().getJMSReplyTo());
             ;
             return retBoolean;
          }

          public void userInitialize()
          {
          }

          public void userTerminate()
          {
          }

       }
```

Once the sample functions to your satisfaction, you can modify the schema to add functionality or create a new schema.

4.1.5. Test HTML Files

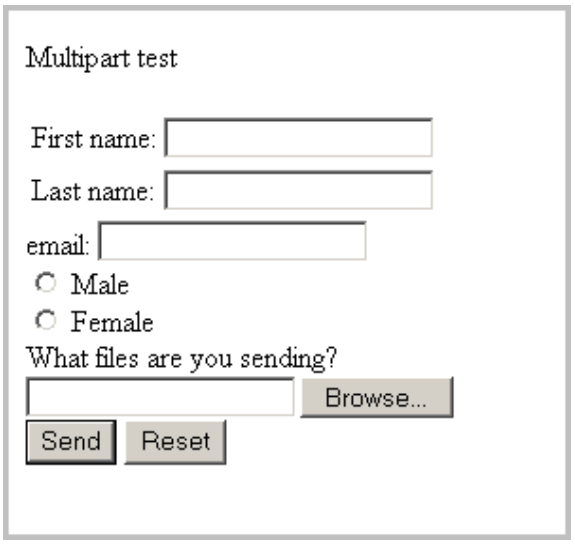
There are two test HTML files provided with the sample:

- testmulptfrm.html
- testurlencoded.html

testmultptfrm.html

The testmultptfrm.html file, for multi-part form data exchange, appears below:

Figure 17 testmultptfrm.html



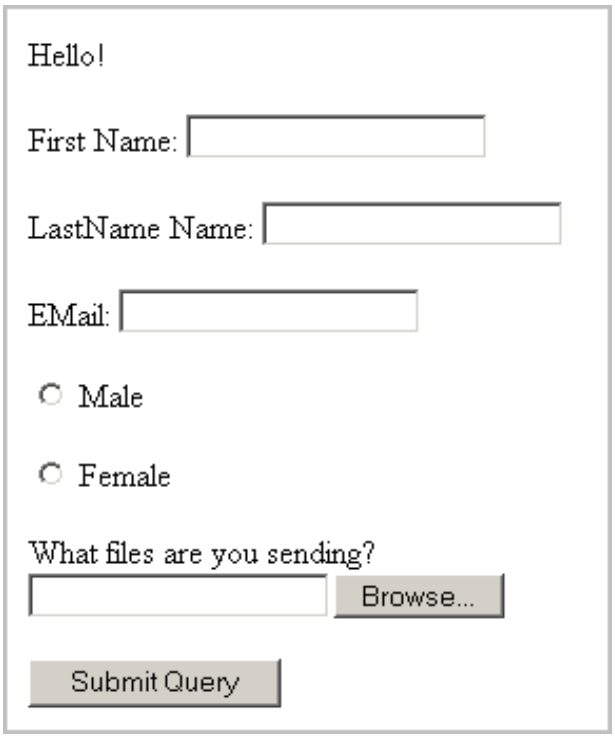
The HTML code follows:

```
<HTML>

<FORM ACTION="cgi-bin/stccgi.exe" METHOD="POST"
ENCTYPE="multipart/form-data">
Multipart test
<P>
<TABLE>
  <TR>
    <TD><LABEL for="fname">First name: </LABEL>
    <TD>  <INPUT type="text" name="firstname" id ="fname">
  <TR>
    <TD><LABEL for="lname">Last name: </LABEL>
    <TD><INPUT type="text" name="lastname" id="lname">
</TABLE>
    <LABEL for="email">email: </LABEL>
              <INPUT type="text" name="email"><BR>
    <INPUT type="radio" name="sex" value="Male"> Male<BR>
    <INPUT type="radio" name="sex" value="Female"> Female<BR>
    <LABEL for="filename">What files are you sending? </LABEL>
             <INPUT type="file" name="filename"><BR>
    <INPUT type="submit" value="Send"> <INPUT type="reset">
    </P>
</FORM>

</HTML>
```

testurlencoded.html

The testurlencoded.html file is provided for name/value pair form data exchange and appears below:

Figure 18 testurlencoded.html



The HTML code appears as follows:

```
<FORM ACTION="cgi-bin/stccgi.exe" METHOD=POST>
Hello!
<P>
First Name:
<INPUT NAME=fname><BR>
<P>
LastName Name:
<INPUT NAME=lname><BR>
<P>
EMail:
<INPUT NAME=email><BR>
<p>
<INPUT type="radio" name="sex" value="Male"> Male<BR>
<p>
<INPUT type="radio" name="sex" value="Female"> Female<BR>
<P>
<LABEL for="filename">What files are you sending? </LABEL>
             <INPUT type="file" name="filename"><BR>
<P>
<INPUT TYPE=submit>
</FORM>
```

4.1.6. Message Routing to Multiple Collaborations

Only one type of topic can be published to each cgi-bin directory. If publishing many topics at the same time is a requirement, multiple cgi-bin directories can be set up. For example, cgi-bin, cgi-bin2, and so on. In each cgi-bin directory, there must be a copy of stccgi.exe, stc_msapi.dll, stc_msclient.dll, stc_mscommon.dll, and mscgi.properties. (For more information see **"Configuring the Web Server Components" on page 23**.)

Modify the mscgi.properties file to point to the correct host and topic/queue to be published.

As an example, if the user wants to publish two topics, Topic:etwebRequestETDTopic and Topic:webRequest, the user modifies the mscgi.properties file, located in cgi-bin, to specify Topic:etwebRequestETDtopic. Topic:webRequest is specified in the mscgi.properties file, located in cgi-bin2. Both cgi-bin and cgi-bin2 directories must contain the same version of the above mentioned files. The user can then create two Collaboration Rules to subscribe to etwebRequestETDTopic and webRequest.

To submit Topic:etwebRequestETDTopic via an HTTP client, the URL format used is:

```
hostname:port\cgi-bin\stccgi.exe
```

To submit Topic:webRequest via an HTTP client, the URL used is:

```
hostname:port\cgi-bin2\stccgi.exe
```

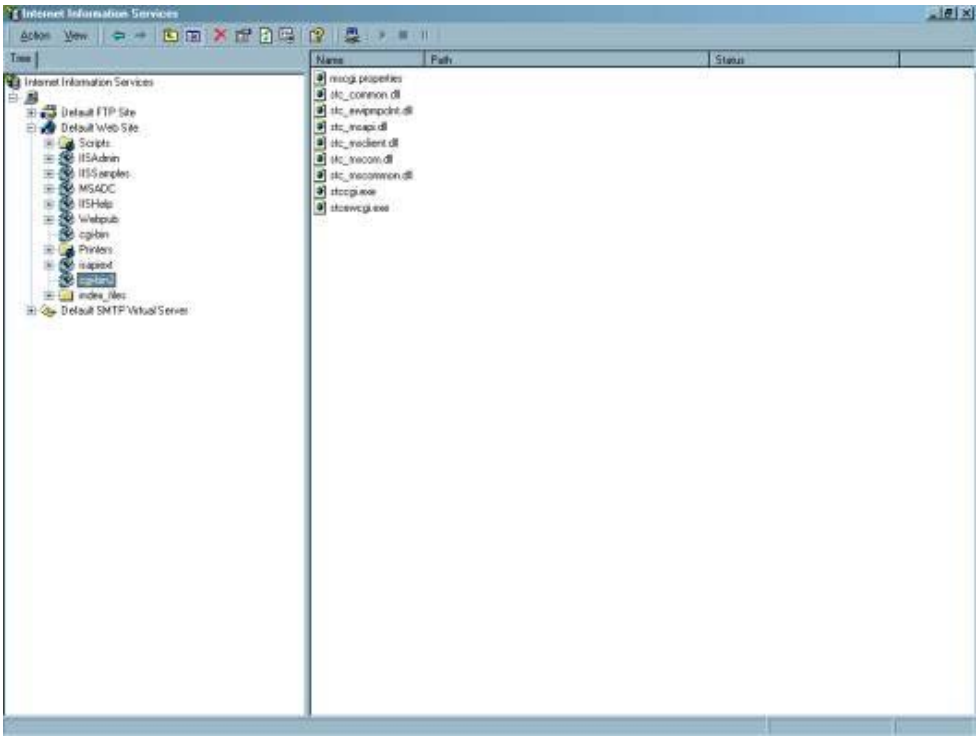
To achieve this result, each Web server must be configured to allow multiple CGI directories.

For Apache, in httpd.conf insert lines equivalent to the following:

```
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache/cgi-bin/"
ScriptAlias /cgi-bin2/ "C:/Program Files/Apache Group/Apache/cgi-
bin2/"
```

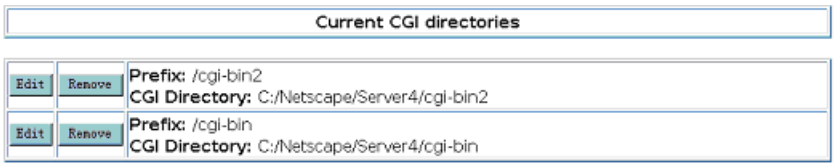
For IIS, create two virtual directories for cgi-bin and cgi-bin2.

Figure 19 Dual IIS Virtual Directories



For iPlanet, configure the server to allow two CGI directories, cgi-bin and cgi-bin2.

Figure 20 iPlanet dual CGI directories



Chapter 5

Methods

5.1 CGI Java e*Way Methods

The Collaboration Editor allows you to call methods by dragging and dropping the node into the Rules dialog box. The node name maybe different from the Java method name. The conversion takes place in the .xsc file. To view the .xsc file, use the ETD Editor.

For example, if the node name is “encoding”, the associated javaName is “Encoding”. If you want to “get” the node value, the Java method called is getEncoding. If you want to “set” the node value, the Java method called is setEncoding.

The CGI Java e*Way components support the following classes:

5.1.1. Class Hierarchy

```
class java.lang.Object
   class com.stc.eways.webETD.Body
   class com.stc.eways.webETD.NameValue
   class com.stc.eways.webETD.OneMimeBodyPart
   class com.stc.eways.webETD.OneMimeBodyPart.NestedMultiPart
   class com.stc.jcsre.SimpleETDImpl (implements com.stc.jcsre.ETD)
      class com.stc.jcsre.MsgETDImpl
           class com.stc.eways.webETD.webReplyETD (implements
             com.stc.jcsre.ETD)
           class com.stc.eways.webETD.webRequestETD (implements
             com.stc.jcsre.ETD)
      class com.stc.eways.webETD.webETDContentType
      class com.stc.eways.webETD.webETDInternetHeaders
      class com.stc.eways.webETD.webReplyETD.ReplyMultiParts
      class com.stc.eways.webETD.webRequestETD.EnvironmentVariables
      class com.stc.eways.webETD.webRequestETD.MultiParts
      class com.stc.eways.webETD.webRequestETD.URLNameValueQueryPairs
```

5.2 Class Body

```
com.stc.eways.webETD
   java.lang.Object
com.stc.eways.webETD.Body
public class Body
extends java.lang.Object
```

The Body Class methods are described in detail on the following pages:

Body

Description

Constructor.

Syntax

```
public Body()
```

getRawPayLoad

Description

Gets the raw payload byte array

Syntax

```
public byte[] getRawPayLoad()
```

Parameters

None.

Returns

byte array
Returns a byte array of the raw payload.

Throws

None.

setRawPayLoad

Description

Sets the raw payload to the byte array

Syntax

```
public void setRawPayLoad(byte[] inrawpayload)
```

Parameters

Name	Type	Description
inrawpayload	byte[]	The raw payload to be set.

Returns

None.

Throws

java.io.IOException

getTransferCodePayLoad

Description

Returns the decoded byte array (Request) or encoded byte array (Reply).

Syntax

```
public byte[] getTransferCodePayLoad()
```

Parameters

None.

Returns

byte array
Returns the decoded or encoded byte array.

Throws

None.

setTransferCodePayLoad

Description

Sets the encoded payload to the byte array.

Syntax

```
public void setTrasferCodePayLoad(byte[] inpayload)
```

Parameters

Name	Type	Description
inpayload	byte[]	The payload to be set.

Returns

None.

Throws

java.io.IOException

getTextStringPayLoad

Description

Gets the charSet decoded string payload.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the decoded string payload.

Throws

None.

setTextStringPayLoad

Description

Sets the charSet encoded string payload.

Syntax

```
public void setTextStringPayLoad(java.lang.String inpayload)
```

Parameters

Name	Type	Description
inpayload	java.lang.String	The payload to be set.

Returns

None.

Throws

None.

marshal

Description

The out bound Collaboration controller calls this marshal method, you only need to either set rawpayload or text string payload.

Syntax

```
public byte[] marshal(java.lang.String encodingstyle,
   java.lang.String charset, boolean isText)
```

Parameters

Name	Type	Description
encodingstyle	java.lang.String	The encoding style.
charset	java.lang.String	The charset.
isText	boolean	true indicates the payload is set to text.

Returns

byte array
Returns a byte array of flattened data.

Throws

com.stc.jcsre.MarshalException

unmarshal

Description

Takes a byte array and parses it into raw payload, decoded payload or string payload based on the content type.

Syntax

```
public void unmarshal(byte[] bytearraybodyin,webETDContentType
contenttype)
```

Parameters

Name	Type	Description
bytearraybodyin	byte[]	An inbound byte array.
contenttype	webETDContentType	The content type to which to parse the data.

Returns

None.

Throws

com.stc.jcsre.UnmarshalException

5.3 Class NameValue

```
java.lang.Object
    com.stc.eways.webETD.NameValue
```

```
public class NameValue
extends java.lang.Object
```

The NameValue Class methods are described in detail on the following pages:

NameValue

Description

Constructor

Syntax

```
public NameValue()
public NameValue(java.lang.String nmin, byte[] valin)
```

Parameters

Name	Type	Description
nmin	java.lang.String	The name.
valin	byte[]	A byte array of value.

getName

Description

Gets the name of the variable.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the name of the variable.

Throws

None.

getValue

Description

Gets the value of this pair.

Syntax

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

Parameters

None.

Returns

byte array
Returns value as a byte array.

Throws

None.

setIsURLencoded

Description

Marks this pair as URL encoded.

Syntax

```
public void setIsURLencoded(boolean isEncoded)
```

Parameters

Name	Type	Description
isEncoded	boolean	true indicates that the part is URL encoded.

Returns

None.

Throws

None.

marshal

Description

Marshal serializes this object into a byte array.

Syntax

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

Parameters

None.

Returns

byte array
Returns the corresponding deserialized NameValue object.

Throws

com.stc.jcsre.MarshalException

unmarshal

Description

Unmarshal deserializes the byte array passed into this object.

Syntax

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

Parameters

Name	Type	Description
inputEvent	byte[]	A byte array deserialized into a NameValue object.

Returns

None.

Throws

com.stc.jcsre.UnmarshalException

5.4 Class OneMimeBodyPart

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.OneMimeBodyPart
public class OneMimeBodyPart
extends java.lang.Object
```

The OneMimeBodyPart Class methods are described in detail on the following pages:

OneMimeBodyPart

Description

Constructor.

Syntax

```
public OneMimeBodyPart()
```

getMimeBodyPart

Description

Returns a javax.mail.internet.MimeBodyPart object.

Syntax

```
public com.stc.eways.webETD.MimeBodyPart getMimeBodyPart()
```

Parameters

None.

Returns

com.stc.eways.webETD.MimeBodyPart
Returns a javax.mail.internet.MimeBodyPart object.

Throws

None.

getMimeBodyPartRawPayLoad

Description

Returns the raw payload byte array.

Syntax

```
public byte[] getMimeBodyPartRawPayLoad()
```

Parameters

None.

Return

byte[]
Returns a byte array of the raw payload.

Throws

javax.mail.MessagingException
java.io.IOException

setMimeBodyPartRawPayLoad

Description

Sets the raw payload.

Syntax

```
public void setMimeBodyPartRawPayLoad(byte[] inbytearray)
```

Parameters

Name	Type	Description
inbytearray	byte[]	A byte array of the raw payload.

Return

None.

Throws

javax.mail.MessagingException
java.io.IOException

getMimeBodyPartDecodedPayLoad

Description

Returns the decoded payload byte array.

Syntax

```
public byte[] getMimeBodyPartDecodedPayLoad()
```

Parameters

None.

Return

byte array
Returns a byte array of the decoded payload.

Throws

javax.mail.MessagingException

java.io.IOException

getMimeBodyPartTextStringPayLoad

Description

Returns the string payload.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the string payload.

Throws

javax.mail.MessagingException
java.io.IOException

setMimeBodyPartTextStringPayLoad

Description

Sets the string payload.

Syntax

```
public void setMimeBodyPartTextStringPayLoad(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The inbound string to set.

Return

None.

Throws

javax.mail.MessagingException
java.io.IOException

isMimeBodyPart

Description

Returns true if the part is a mime part, or false if it is URL encoded.

Syntax

```
public boolean isMimeBodyPart()
```

Parameters

None.

Returns

boolean
Returns true if the part is a mime part, otherwise, returns false.

Throws

None.

isMimeType

Description

Queries whether this part of the specified MIME type. This method compares only the primaryType and subType.

Syntax

```
public boolean isMimeType(java.lang.String typein)
```

Parameters

Name	Type	Description
typein	java.lang.String	The part to query.

Returns

boolean
Returns true if the part is a MIME type.

Throws

javax.mail.MessagingException

getDisposition

Description

Returns the content disposition parameter as a string.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the content disposition parameter as a string.

Throws

javax.mail.MessagingException

setDisposition

Description

Sets the disposition of the content type parameter.

Syntax

```
public void setDisposition(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The content type to be set.

Return

None.

Throws

javax.mail.MessagingException

setContentID

Description

Sets the ContentID content type parameter.

Syntax

```
public void setContentID(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The contentID to be set.

Return

None.

Throws

javax.mail.MessagingException

getContentID

Description

Gets the contentID content type parameter.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the contentID content type parameter.

Throws

javax.mail.MessagingException

getContentTypeString

Description

Returns the contentType string.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the contentType string.

Throws

javax.mail.MessagingException

setContentMD5

Description

Sets the contentMD5 for this body part.

Syntax

```
public void setContentMD5(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The contentMD5 to be set.

Return

None.

Throws

javax.mail.MessagingException

getContentMD5

Description

Gets the contentMD5 for this body part.

Syntax

```
public java.lang.String getContentMD5()
```

Parameters

None.

Return

java.lang.String
Returns the contentMD5 for this body part.

Throws

javax.mail.MessagingException

getDescription

Description

Gets the description of this body part.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the description for the body part.

Throws

javax.mail.MessagingException

setDescription

Description

Sets the description of this body part.

Syntax

```
public void setDescription(java.lang.String instr)
   throws javax.mail.MessagingException
```

Parameters

Name	Type	Description
instr	java.lang.String	The description to be set.

Return

None.

Throws

javax.mail.MessagingException

getEncoding

Description

Gets the encoding of this body part.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the encoding for the body part.

Throws

javax.mail.MessagingException

getFileName

Description

Gets the filename parameter for this body part.

Syntax

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

Parameters

None.

Return

java.lang.String
Returns the filename parameter for the body part.

Throws

javax.mail.MessagingException

---

setFileName

Description

Sets the filename of this body part.

Syntax

```
public void setFileName(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The filename to be set.

Return

None.

Throws

javax.mail.MessagingException

---

getContentType

Description

Gets the content type as a webETDContentType object for this body part.

Syntax

```
public webETDContentType getContentType()
```

Parameters

None.

Return

webETDContentType
Returns the content type as a webETDContentType object for the body part.

Throws

javax.mail.MessagingException

countNestedMultiPart

Description

Counts the number of parts in a nested multipart.

Syntax

```
public int countNestedMultiPart()
```

Parameters

None.

Return

int
Returns the number of parts in the nested multipart form, if none, returns zero.

Throws

None.

getNestedMultiPart

Description

Returns the nested multipart.

Syntax

```
public OneMimeBodyPart getNestedMultiPart(int index)
```

Parameters

Name	Type	Description
index	int	The index number for the nested multipart to be returned.

Return

OneMimeBodyPart
Returns the nested multipart object.

Throws

None.

marshal

Description

Serializes this object into a byte array.

Syntax

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

Parameters

Name	Type	Description
buildmbponly	boolean	The object to be deserialized. The build MimeBodyPart does not return a byte array.

Return

byte[]
Returns a byte array corresponding to a deserialized OneMimeBodyPart object, else null.

Throws

com.stc.jcsre.MarshalException
javax.mail.MessagingException
java.io.IOException

unmarshal

Description

Deserializes the byte array passed into this object.

Syntax

```
public void unmarshal(byte[] abodyins)
   throws javax.mail.MessagingException,
          java.io.IOException
```

Parameters

Name	Type	Description
abodyins	byte[]	The byte array to be deserialized into a OneMimeBodyPart object.

Return

None.

Throws

javax.mail.MessagingException
java.io.IOException

5.5 Class OneMimeBodyPart.NestedMultiPart

```
com.stc.eways.webETD
Class OneMimeBodyPart.NestedMultiPart
java.lang.Object
   com.stc.eways.webETD.OneMimeBodyPart.NestedMultiPart
```

```
Enclosing class:OneMimeBodyPart
public class OneMimeBodyPart.NestedMultiPart
extends java.lang.Object
```

The OneMimeBodyPart.NestedMultiPart Class methods are described in detail on the following pages:

OneMimeBodyPart.NestedMultiPart

Description

Constructor.

Syntax

```
public OneMimeBodyPart.NestedMultiPart()
```

setCurrIndex

Description

Sets the current index point.

Syntax

```
public void setCurrIndex(int i)
```

Parameters

Name	Type	Description
i	int	The index number for the nested multpart to be set.

Return

None.

Throws

None.

getSize

Description

Gets the size.

Syntax

```
public int getSize()
```

Parameters

None.

Return

int
Returns the size of the object.

Throws

None.

---

getPart

Description

Gets the part.

Syntax

```
public OneMimeBodyPart getPart()
```

Parameters

None.

Return

OneMimeBodyPart
Returns the OneMimeBodyPart object.

Throws

None.

---

unmarshal

Description

Deserializes the byte array passed into this object.

Syntax

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

Parameters

Name	Type	Description
bytearraybodyin	byte[]	The byte array to be deserialized into a OneMimeBodyPart object.

Return

None.

Throws

com.stc.jcsre.UnmarshalException
javax.mail.MessagingException
java.io.IOException

5.6 Class webReplyETD

```
java.lang.Object
   com.stc.jcsre.StimpleETDImpl
       com.stc.jcsre.MsgETDImpl
           com.stc.eways.webETD.webReplyETD
All Implemented Interfaces: com.stc.jcsre.ETD,
com.stc.jcsre.ETDConstansts
public class webReplyETD
extends com.stc.jcsre.MsgETDImpl
implements com.stc.jcsre.ETD
```

The webReplyETD Class methods are described in detail on the following pages:

webReplyETD

Description

Constructor.

Syntax

```
public webReplyETD()
```

getJMSReplyTo

Description

Gets the JMSReplyTo field. This is normally obtained from the corresponding webRequestETD.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the JMSReplyTo field.

Throws

None.

setJMSReplyTo

Description

Sets the JMSReplyTo filed. This is normally obtained from the corresponding webRequestETD.

Syntax

```
public void setJMSReplyTo(java.lang.String replyTo)
```

Parameters

Name	Type	Description
replyTo	java.lang.String	The JMSReplyTo field to set.

Returns

None.

Throws

None.

send

Description

Overrides the super's send( ) method to work with JMS. You must call setJMSReplyTo before call this method.

Syntax

```
public void send()
```

Parameters

None.

Return

None.

Throws

None.

Specified by

send in interface `com.stc.jcsre.ETD`

Overrides

send in class `com.stc.jcsre.MsgETDImpl`

---

getBody

Description

Gets the body of this reply, if the reply is of a discrete type (text/plain, image/jpeg).

Syntax

```
public Body getBody()
```

Parameters

None.

Return

Body
Returns the body of this reply if the reply is discrete type. For example, text/plain or image/jpeg.

Throws

None.

---

getContentType

Description

Gets the content type of this reply, as a webETDContentType object.

Syntax

```
public webETDContentType getContentType()
```

Parameters

None.

Returns

webETDContentType
Returns the webETDContentType object.

Throws

None.

addHeader

Description

Adds a name/value pair as a new header to this reply.

Syntax

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

Parameters

Name	Type	Description
name	java.lang.String	The name string
value	java.lang.String	The value string

Returns

None.

Throws

None.

setHeader

Description

Sets the name/value pair header. If the header name already exists, the associated value is overwritten.

Syntax

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

Parameters

Name	Type	Description
name	java.lang.String	The name of the header to set.
value	java.lang.String	The value to set for the header.

Returns

None.

Throws

None.

getHeader

Description

Gets the first header line by the name and delimiter.

Syntax

```
public java.lang.String getHeader(java.lang.String name,
java.lang.String delim)
```

Parameters

Name	Type	Description
name	java.lang.String	The name of the header to get.
delim	java.lang.String	The delimiter.

Returns

java.lang.String
Returns the first header line value.

Throws

None.

getMultiParts

Description

Gets the value for the indexed part, provided it is multipart data.

Syntax

```
public OneMimeBodyPart getMultiParts(int index)
```

Parameters

Name	Type	Description
index	int	The index point for the part.

Returns

OneMimeBodyPart
Returns the value for the part.

Throws

None.

isMultipart

Description

Queries whether the request, originated as multipart data.

Syntax

```
public boolean isMultipart()
```

Parameters

None.

Returns

boolean
Returns true if the request originated as multipart data.

Throws

None.

---

countMultiParts

Description

Counts the number of parts contained in the multipart body.

Syntax

```
public int countMultiParts()
```

Parameters

None.

Returns

int
Returns the number of parts.

Throws

None.

---

marshal

Description

Serializes the object into a byte array.

Syntax

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

Parameters

None.

Returns

byte array
Returns a byte array of the serialized object.

Throws

com.stc.jcsre.MarshalException

Specified by

marshal in interface com.stc.jcsre.ETD

Overrides

`marshal` in class `com.stc.jcsre.SimpleETDImpl`

5.7 Class webRequestETD

```
java.lang.Object
   com.stc.jcsre.SimpleETDImpl
       com.stc.jcsre.MsgETDImpl
           com.stc.eways.webETD.webRequestETD
All Implemented Interfaces:
com.stc.jcsre.ETD, com.stc.jcsre.ETDConstants
public class webRequestETD
extends com.stc.jcsre.MsgETDImmpl
implements com.stc.jcsre.ETD
```

The webRequestETD Class methods are described in detail on the following pages:

webRequestETD

Description

Constructor.

Syntax

```
public webRequestETD()
```

getJMSReplyTo

Description

Gets the JMSReplyTo for this Event.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns JMSReplyTo string.

Throws

None.

getContentType

Description

Gets the ContentType object.

Syntax

```
public webETDContentType getContentType()
```

Parameters

None.

Returns

webETDContentType
Returns webETDContentType object.

Throws

None.

getRequestMethod

Description

Gets request method string.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the request method string.

Throws

None.

getEnvironmentVariables

Description

Gets the environment variables object.

Syntax

```
public webRequestETD.EnvironmentVariables getEnvironmentVariables(int
index)
```

Parameters

Name	Type	Description
index	int	The index point.

Returns

webRequestETD.EnvirnmentVariables
Returns the webRequestETD.EnvironmentVariable object.

Throws

None.

countEnvironmentVariables

Description

Counts the number of environment variables that are part of the JMS property passed in. Each environment variable is a name value pair. For example, SERVER_PROTOCOL=HTTP/1.1.

Syntax

```
public int countEnvironmentVariables()
```

Parameters

None.

Returns

int
Returns the number of environment variables.

Throws

None.

getURLNameValueQueryPairs

Description

Gets the specified name/value pair, URL decoded.

Syntax

```
public webRequestETD.URLNameValueQueryPairs
getURLNameValueQueryPairs(int index)
```

Parameters

Name	Type	Description
index	int	The index point.

Returns

webRequestETD.URLNameValueQueryPairs
Returns the specified name/value pair.

Throws

None.

isSingleBody

Description

Queries whether the request originated as Application/x-www-urlencoded data.

Syntax

```
public boolean isSingleBody()
```

Parameters

None.

Returns

boolean
Returns true if the original request was sent in Application/x-www-urlencoded format.

Throws

None.

getBody

Description

Gets the body of this webRequestETD, provided the content type is discrete (text/plain or image/jpeg).

Syntax

```
public Body getBody()
```

Parameters

None.

Returns

Body
Returns the body of the webRequestETD.

Throws

None.

isUrlencoded

Description

Queries whether the HTTP request (GET) is URL encoded.

Syntax

```
public boolean isUrlencoded()
```

Parameters

None.

Return Value

boolean
Returns true to indicate that the HTTP request (GET) is URL encoded.

Throws

None.

countURLNameValueQueryPairs

Description

Counts the number of environment variables.

Syntax

```
public int countURLNameValueQueryPairs()
```

Parameters

None.

Return Value

int
Returns the number of environment variables.

Throws

None.

getMultiParts

Description

Gets the specified part.

Syntax

```
public OneMimeBodyPart getMultiParts(int index)
```

Parameters

Name	Type	Description
index	int	The index point.

Returns

OneMimeBodyPart
Returns the OneMimeBodyPart object.

Throws

None.

isMultipart

Description

Queries whether the request originated as Multipart/form-data.

Syntax

```
public boolean isMultipart()
```

Parameters

None.

Returns

boolean
Returns true if the request originated as Multipart/form-data.

Throws

None.

countMultiParts

Description

Counts the number of MIME parts.

Syntax

```
public int countMultiParts()
```

Parameters

None.

Returns

int

Returns the number of MIME parts.

Throws

None.

---

unmarshal

Description

Deserializes the byte array passed into this object.

Syntax

```
public void unmarshal(byte[] inputEvent)
   throws com.stc.jcsre.UnmarshalException
```

Parameters

Name	Type	Description
inputEvent	byte array	The input Event to be deserialized.

Returns

None.

Throws

com.stc.jcsre.MarshalException

Specified by

`unmarshal` in interface `com.stc.jcsre.ETD`

Overrides

`unmarshal` in class `com.stc.jcsre.SimpleETDImpl`

5.8 Class webETDContentType

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.webETDContentType
public class webETDContentType
extends java.lang.Object
```

The webETD Class methods are described in detail on the following pages:

webETDContentType on page 89

setCharSet on page 92

webETDContentType

Description

Constructor.

Syntax

```
public webETDContentType()
```

unmarshal

Description

Deserializes an object input put as a string.

Syntax

```
public void unmarshal(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The input as a string.

Returns

None.

Throws

javax.mail.internet.ParseException

toString

Description

Returns a string representation of the object.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns a string representation of the object.

Throws

None.

Overrides

`toString` in class `java.lang.Object`

---

getPrimaryType

Description

Gets the primary type in a media type, such as, text in text/plain.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the primary type as a string.

Throws

None.

---

setPrimaryType

Description

Sets the primary type in the object. For example, text in text/plain.

Syntax

```
public void setPrimaryType(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The primary type to set.

Return Value

None.

Throws

None.

getSubType

Description

Gets the sub type in media type. For example, plain in text/plain.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the sub type in a media type.

Throws

None.

setSubType

Description

Sets the sub type in the object. For example, plain in text/plain.

Syntax

```
public void setSubType(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The sub type to set.

Returns

None.

Throws

None.

getCharSet

Description

Gets the charSet in a media type. For example, EUC_JP.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the charSet value.

Throws

None.

setCharSet

Description

Sets the charSet in a media type. For example, EUC_JP.

Syntax

```
public void setCharSet(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The charSet type to set.

Returns

None.

Throws

None.

getMultipartBoundary

Description

Gets the boundary parameter for a content type.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the boundary parameter.

Throws

None.

setMultipartBoundary

Description

Sets the boundary parameter in a content type.

Syntax

```
public void setMultipartBoundary(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The boundary to set.

Returns

None.

Throws

None.

getContentTransferEncoding

Description

Gets the content transfer encoding parameter for a content type. For example, base64.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the content transfer encoding parameter.

Throws

None.

setContentTransferEncoding

Description

Sets the content transfer encoding parameter in a content type. For example, base64.

Syntax

```
public void setContentTransferEncoding(java.lang.String instr)
```

Parameters

Name	Type	Description
instr	java.lang.String	The content transfer encoding to set.

Returns

None.

Throws

None.

getContentTypeParameter

Description

Gets a specified parameter in a content type. For example, ContentID.

Syntax

```
public java.lang.String getContentTypeParameter(java.lang.String nm)
```

Parameters

Name	Type	Description
nm	java.lang.String	The name of the parameter to get.

Returns

java.lang.String
Returns a parameter as a string.

Throws

None.

setContentTypeParameter

Description

Sets the parameter value in a content type. For example, ContentID.

Syntax

```
public void setContentTypeParameter(java.lang.String nm,
                                    java.lang.String val)
```

Parameters

Name	Type	Description
nm	java.lang.String	The name of the parameter to set.
val	java.lang.String	The value to be set.

Returns

None.

Throws

None.

marshal

Description

Serializes the object into a byte array.

Syntax

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

Parameters

None.

Returns

byte array
Returns a byte array of the serialized object.

Throws

com.stc.jcsre.MarshalException

5.9 Class webETDInternetHeaders

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.webETDInternetHeaders
public class webETDInternetHeaders
extends java.lang.Object
```

webETDInternetHeaders

Description

Constructors.

Syntax

```
public webETDInternetHeaders()
```

addHeader

Description

Adds a name/value pair header. The delimiter is set by default.

Syntax

```
public void addHeader(java.lang.String nm,
                      java.lang.String val)
```

Parameters

Name	Type	Description
nm	java.lang.String	The name of the parameter to set.
val	java.lang.String	The value to be set.

Returns

None.

Throws

None.

setHeader

Description

Sets a name/value pair header delimiter. If the header already exists, it is overwritten.

Syntax

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

Parameters

Name	Type	Description
name	java.lang.String	The name of the parameter to set.
value	java.lang.String	The value to be set.

Returns

None.

Throws

None.

getHeader

Description

Gets the value of a header and delimiter. If multiple header lines match, the first one is returned.

Syntax

```
public java.lang.String getHeader(java.lang.String nm,
                                  java.lang.String delim)
```

Parameters

Name	Type	Description
nm	java.lang.String	The name of the parameter to get.
delim	java.lang.String	The delimiter to get.

Returns

java.lang.String
Returns the value of a header and delimiter. If multiple header lines match, the first one is returned.

Throws

None.

getAsInternetHeaders

Description

Gets the headers as internet headers.

Syntax

```
public javax.mail.internet.InternetHeaders getAsInternetHeaders()
```

Parameters

None.

Returns

javax.mail.internet
Returns the headers as javax.mail.internet.InternetHeaders.

Throws

None.

marshal

Description

Serializes the object into a byte array.

Syntax

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

Parameters

None.

Returns

byte array
Returns a byte array of the serialized object.

Throws

com.stc.jcsre.MarshalException

5.10 Class webReplyETD.ReplyMultiParts

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.webReplyETD.ReplyMultiParts
Enclosing class:
webReplyETD
   public class webReplyETD.ReplyMultiParts
extends java.lang.Object
```

The webReplyETD.ReplyMultiParts Class methods are described in detail on the following pages:

webReplyETD.ReplyMultiParts

Description

Constructor.

Syntax

```
public webReplyETD.ReplyMultiParts()
```

setCurrIndex

Description

Sets the current index.

Syntax

```
public void setCurrIndex(int i)
```

Parameters

Name	Type	Description
i	int	The index point to set.

Returns

None.

Throws

None.

getSize

Description

Gets the size of the part.

Syntax

```
public int getSize()
```

Parameters

None.

Returns

int
Returns the size of the part.

Throws

None.

getPart

Description

Gets the OneMimeBodyPart object.

Syntax

```
public OneMimeBodyPart getPart()
```

Parameters

None.

Returns

OneMimeBodyPart
Returns the OneMimeBodyPart object.

Throws

None.

marshal

Description

Serializes the object into a byte array, based on the content type.

Syntax

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

Parameters

Name	Type	Description
topcontenttype	webETDContentType	The webETDContentType object.

Returns

byte array
Returns a byte array of the serialized object.

Throws

com.stc.jcsre.MarshalException

5.11 Class webRequestETD.EnvironmentVariables

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.webRequestETD.EnvironmentVariables
Enclosing class:
   webRequestETD
public class webRequestETD.EnvironmentVariables
extends java.lang.Object
```

The webRequestETD.EnvironmentVariables Class methods are described in detail on the following pages:

webRequestETD.EnvironmentVariables

Description

Constructor.

Syntax

```
public webRequestETD.EnvironmentVariables()
```

setCurrIndex

Description

Sets the current index point.

Syntax

```
public void setCurrIndex(int i)
```

Parameters

Name	Type	Description
i	int	The index point to set.

Returns

None.

Throws

None.

addEnvVarPair

Description

Adds an Environmental variable pair.

Syntax

```
protected void addEnvVarPair(NameValue newpair)
```

Parameters

Name	Type	Description
newpair	NameValue	The name value object to be added.

Returns

None.

Throws

None.

getSize

Description

Gets the size of the variable.

Syntax

```
public int getSize()
```

Parameters

None.

Returns

int
Returns the size of the variable.

Throws

None.

---

getName

Description

Gets the name of the variable.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the name of this environment variable.

Throws

None.

---

getValue

Description

Gets the value of the variable.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the value for the environment variable.

Throws

None.

5.12 Class webRequestETD.MultiParts

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.webRequestETD.MultiParts
Enclosing class:
webRequestETD
public class webRequestETD.MultiParts
extends java.lang.Object
```

The webRequestETD.MultiParts Class methods are described in detail on the following pages:

webRequestETD.MultiParts

Description

Constructor.

Syntax

```
public webRequestETD.MultiParts()
```

setCurrIndex

Description

Sets the current index point.

Syntax

```
public void setCurrIndex(int i)
```

Parameters

Name	Type	Description
i	int	The index point to set.

Returns

None.

Throws

None.

getSize

Description

Gets the size of the part.

Syntax

```
public int getSize()
```

Parameters

None.

Returns

int
Returns the size of the part.

Throws

None.

unmarshal

Description

Deserializes the byte array passed into this object.

Syntax

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

Parameters

Name	Type	Description
bytearraybodyin	byte array	The object to be deserialized.

Returns

None.

Throws

com.stc.jcsre.UnmarshalException
java.io.IOException
javax.mail.MessagingException

getPart

Description

Gets the part.

Syntax

```
public OneMimeBodyPart getPart()
```

Parameters

None.

Returns

OneMimeBodyPart
Returns the OneMimeBodyPart object.

Throws

None.

5.13 Class webRequestETD.URLNameValueQueryPairs

```
com.stc.eways.webETD
java.lang.Object
   com.stc.eways.webETD.webRequestETD.URLNameValueQueryPairs
Enclosing class:
webRequestETD
public class webRequestETD.URLNameValueQueryPairs
extends java.lang.Object
```

The webRequestETD.IURLNameValueQueryPairs Class methods are described in detail on the following pages:

webRequestETD.URLNameValueQueryPairs

Description

Constructor.

Syntax

```
public webRequestETD.URLNameValueQueryPairs()
```

getSize

Description

Gets the size of the name/value pair.

Syntax

```
public int getSize()
```

Parameters

None.

Returns

int
Returns the size of the name/value pair.

Throws

None.

getName

Description

Gets the name of the name/value pair.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the name of the name/value pair.

Throws

None.

getValue

Description

Gets the value for the name/value pair.

Syntax

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

Parameters

None.

Returns

java.lang.String
Returns the value for the name/value pair.

Throws

None.

setCurrIndex

Description

Sets the current index point.

Syntax

```
public void setCurrIndex(int i)
```

Parameters

Name	Type	Description
i	int	The index point to set.

Returns

None.

Throws

None.

unmarshal

Description

Separates the URL encoded string into name/value pairs.

Syntax

```
public void unmarshal(java.lang.String urlencodedstring)
```

Parameters

Name	Type	Description
urlencodedstring	java.lang.String	The string to be broken down into a name/value pair.

Returns

None.

Throws

com.stc.jcsre.UnmarshalException

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W