SeeBeyond ICAN Suite

e*Way Intelligent Adapter for iPlanet Application Server User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Contents

Chapter 4

Chapter 5

Chapter 1

Introduction

This chapter describes how to install the e*Way Intelligent Adapter for the iPlanet Application Server (iPlanet App Server), as well as how to create the Java code required to enable the application server to communicate with e*Gate.

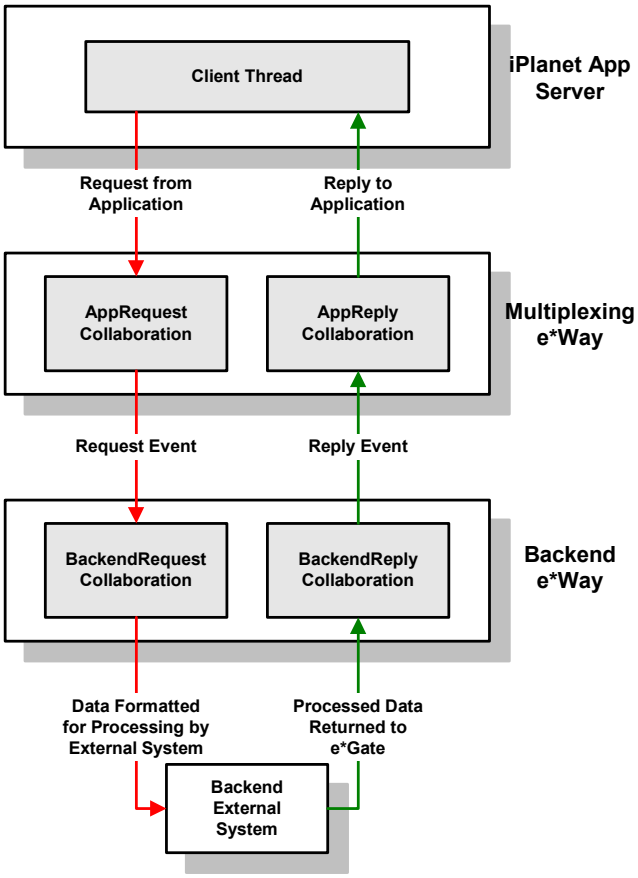
1.1 Overview

The iPlanet Application Server uses Hypertext Transfer Protocol (HTTP) to deliver World Wide Web documents to clients using internet browsers such as Microsoft Internet Explorer or Netscape Navigator. The iPlanet e*Way extends the functionality of the web server by making external data sources available to calls from within Java servlets and Java Server Pages (JSP files).

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 moderate to advanced-level knowledge of operations and administration for the operating system(s) under which the iPlanet Application Server and e*Gate systems run; to be thoroughly familiar with Java scripting; and to be thoroughly familiar with Windows-style GUI operations.

Figure 1 Overview of the iPlanet e*Way implementation



1.1.2 Components

The iPlanet e*Way is comprised of the following:

- iPlanet application server components: Java servlets, JSP files, extension packages, and supporting library files.
- Participating Host components: a multiplexing e*Way and supporting library files.

A complete list of installed files appears in **Table 1 on page 12** and **Table 2 on page 12**.

1.2 Supported Operating Systems

The iPlanet Application Server e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0 and 11i (PA-RISC)
- Sun Solaris 8 and 9

1.3 System Requirements

To use the iPlanet Application Server e*Way, you need the following:

- An e*Gate Participating Host.
- A TCP/IP network connection.
- A computer running Windows, to allow you to use the e*Gate Schema Designer and ETD Editor
- Additional disk space for e*Way executable, configuration, library, and script files. The disk space is required on both the Participating and the Registry Host. Additional disk space is required to process and queue the data that this e*Way processes. The amount necessary varies based on the type and size of the data being processed, and any external applications performing the processing.

For the iPlanet Application Server that will communicate with the iPlanet e*Way, you need a client system with the following:

- iPlanet Application Server version 6.0
- Sufficient memory and disk space to support web server functions. See your iPlanet Application Server User's Guides for more information about server requirements.

Installed on the Participating Host

- Java JDK version 1.3.1

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

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 requirements for installing the iPlanet e*Way and how to configure the iPlanet App Server components needed. A list of the files and directories created by the installation is also provided.

2.1 Installing the iPlanet e*Way

The iPlanet e*Way must be installed on the e*Gate Participating Host machine. The **stcph.jar** file (and any desired samples) must be copied onto the iPlanet App Server machine (if different from the Participating Host). (See **Table 2 on page 12** for details.)

2.1.1 Pre-installation

- Exit all programs before running the setup program, including any anti-virus applications.
- You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.

2.1.2 Installation Procedure

If you are installing this e*Way as part of a complete e*Gate installation, please follow the instructions in the *e*Gate Integrator Installation Guide*. The iPlanet e*Way is installed as an "Add-on" component in the fourth phase of the installation.

If you are adding the iPlanet e*Way to an existing e*Gate installation, follow the instructions below.

To install the iPlanet e*Way Components on Windows systems

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
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**. Click **Next** as necessary to proceed through the setup application.

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

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

7 Select the e*Way line, and then click the "**Change**" button to edit which e*Ways are installed from the provided CD, or select the e*Way line to install all e*Ways on the CD.

8 When the **Select Components** dialog box appears, check **iPlanet App Server e*Way.**

9 When the **Registry Hostname** window appears, enter the name of the machine on which the Registry files are installed.

10 Complete the fields in the **Administrator Account Information** dialog box.

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

11 Follow the on-screen prompts to complete the installation.

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

To install the iPlanet e*Way on a UNIX system:

1 Log on to the workstation on which you want to install the e*Way. If you do not wish to log in as root, you must log in as a user with sufficient privilege to install files in the "egate" directory tree.

2 Insert the CD-ROM into the drive.

3 If necessary, mount the CD-ROM drive. On HP-UX systems, you must mount the drive with this command:

/etc/mount -F cdfs -o cdcase /dev/cdrom

where **/cdrom** is the mount point.

4 At the shell prompt, type:

cd /cdrom/setup

5 Start the installation script by typing:

setup.sh

6 If you are not running as root, you will see a message notifying you that services will not start automatically for non-root users. Press **Enter** to continue.

7 You will see a message confirming that you are running the e*Gate installation script, and reminding you that you can type **-** (hyphen) to back up a step or **QUIT** (all capitals) to exit the install program. Press **Enter** to continue.

8 You will be prompted to accept the license agreement. Type **y** and press **Enter**.

The platform type and a menu of options will display:

```
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
```

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

10 You will be prompted for the installation path. Press **Enter** to accept the default path, or type a new path and press **Enter**.

- If you are logged in as root, the suggested path will be **/opt/egate/client**.
- If you are logged in under any other user name, the suggested path will be **/home/*username*/egate/client**.

Whether you install e*Gate to a **/home** directory to an application directory such as **/opt**, we strongly recommend that you use the recommended relative path "**egate/client**" as the destination directory for the add-on-application installation.

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

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

If you selected "**U**," you will see a warning regarding shared EXE and DLL files. Read this warning and press **Enter** to continue.

12 Enter the name of the Registry Server supporting these add-on applications. If the installation utility detects a Registry Host running on the current server, it will suggest that host's name.

13 You will be prompted 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 will appear. Type the number corresponding to the add-on package(s) you wish to install (**e*Ways**) and press **Enter**.

16 A menu of e*Ways will appear. Type the number corresponding to the e*Way you wish to install and press **Enter**.

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

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

19 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.**
- Select another option to continue the installation.

2.1.3 Configuring the Participating Host Components in the Schema Designer

Important

From the perspective of the e*Gate GUIs, the iPlanet e*Way is not a system of components distributed between the iPlanet App Server and a Participating Host, but a single component that runs an executable file (the multiplexer **stcewipmp.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 iPlanet e*Way system.

To configure the Participating Host components:

1. If you have not already done so, launch the Schema Designer.
2. Using the Component editor, create a new e*Way.
3. Display the new e*Way's properties.
4. On the General tab, under **Executable File**, click **Find**.
5. Select the file **stcewipmp.exe**.
6. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 3**. The setup and requirements of schemas required to use this e*Way are discussed in **Chapter 5**.

Note: *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the online Help system.*

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

2.2 Files/Directories Created by the Installation

The iPlanet e*Way installation process installs the following files in the e*Gate directory tree. The files located in Table 1 are installed in the "egate\client" tree. The files located in Table 2 need to be copied from the Participating Host system onto the iPlanet App Server machine. These files must be committed to the "default" schema on the Registry Host.

Table 1 Participating Host System files installed

e*Gate Directory	File(s)
bin\	stcewipmp.exe stc_common.dll stcewipmpclnt.dll
configs\stcewipmp\	stcewipmp.def
classes\	stcph.jar

Table 2 lists the files that must be used to create a Web Application Module. For more information, please refer to the iPlanet Application Server documentation.

Table 2 iPlanet Application Server files

Web Server Directory	File(s)
classes\	stcph.jar
%ServletDirectory%	SampleServlet.class

Note: *The user-defined files must be included in the iPlanet Application package.*

The files in Table 2 must be copied to the iPlanet Application Server Machine. The iPlanet Application Server classpath must include **stcph.jar**. Please refer to the iPlanet Application Server documentation for more information regarding the recommended location to which to copy this file.

Once the files have successfully been copied onto the iPlanet Application Server and any applicable classpath adjustments have been made, the iPlanet Application Server must be stopped and restarted to ensure that the new class information is recognized.

For more information about configuring the iPlanet Application Server see:

http://docs.iplanet.com/docs/manuals/fasttrak/41/ag/esprgrm.htm

Chapter 3

Configuration

Important: *From the perspective of the e*Gate GUIs, the iPlanet 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 multiplexer* ***stcewipmp.exe****). When this manual discusses procedures within the context of any e*Gate GUI (such as those in this chapter, which deals in part with the e*Way Editor), the term "e*Way" refers only to the Participating Host component of the iPlanet e*Way system.*

3.1 e*Way Configuration Parameters

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

To change e*Way configuration parameters:

1. In the Schema Designer's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
3. In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *e*Gate Integrator User's Guide*.

The e*Way's configuration parameters are organized into a single section: General Settings.

3.1.1 General Settings

The parameters in this section specify the name of the external client system and the IP port through which e*Gate and the client system communicate.

Request Reply IP Port

Description

Specifies the IP port that the e*Way will listen (bind) for client connections. This parameter is used for Request/Reply behavior.

Required Values

A valid TCP/IP port number between 1 and 65536. The default is **26051**. Normally, you only need to change the default number if the specified TCP/IP port is in use, or you have other requirements for a specific port number.

Push IP Port

Description

Specifies the IP port through which this e*Way allows an external system to connect and receive unsolicited (without submitting a request) Events.

Required Values

A valid TCP/IP port number between 0 and 65536. The default is **0**.

Additional Information

Any Event that this e*Way receives that has zero values for all fields in the 24 byte MUX header is sent to all callers of the **WaitForUnsolicited**. This parameter is optional. If set to zero, the e*Way will follow the Request/Reply scenario and not accept unsolicited Events.

Rollback if no Clients on Push Port

Description

Specifies whether the Event will continually roll back if there are no push clients connected.

Required Values

Yes or **No**. If set to **Yes**, the Event will continually roll back if there are no push clients connected.

Wait For IQ Ack

Description

Specifies whether the send client function does NOT return until the Event is committed to the IQ.

Required Values

Yes or **No**. If set to **Yes**, the send client function does NOT return until the Event is committed to the IQ.

Caution: *This parameter should be set if the data must be committed to the IQ on every transaction before the API returns to the client. Setting this parameter to Yes will*

significantly impact performance. If normal request/reply type transactions are being sent/received, and the data can be recreated at the client, this parameter should not be set.

Send Empty MSG When External Disconnect

Description

Specifies whether the e*Way sends an empty incoming message (containing only the multiplexer header) when an external client disconnects.

Required Values

Yes or **No**. If set to **Yes**, the e*Way sends an empty incoming message when an external client disconnects.

MUX Instance ID

Description

Specifies whether the specified 8 (eight) bytes are prepended to the 24 (twenty-four) byte session ID of the request received from the external connection before sending to e*Gate.

Required Values

A string. If this value is other than "0", the 8 bytes are prepended to the 24 byte session ID. The default is **0**.

Note: *This is a string where "00" and "00000000" are valid MUX Instance IDs, while "0" is to turn this option off. Only the first 8 bytes are used.*

MUX Recovery ID

Description

Specifies whether the 8 bytes are prepended to the reply and republish back to e*Gate provided the value is other than "0" and the multiplexer finds that the session related to the MUX ID in the return message has been dropped.

Required Values

A string. If this value is other than "0", the 8 bytes are prepended to the 24 byte session ID. The default is **0**.

Note: *This is a string where "00" and "00000000" are valid MUX Recovery IDs, while "0" is to turn this option off. Only the first 8 bytes are used.*

3.2 External Configuration Requirements

To enable the client system to communicate with the e*Gate API Kit, you must do the following:

1. Install the required client files on the external system (see **"Files/Directories Created by the Installation" on page 12**).
2. Configure the client components as necessary to use the TCP/IP port specified above in **"Push IP Port" on page 14**.

Chapter 4

APIs

4.1 com.stc.MUXPooler

The **MUXPooler** class operates between the multi-plexing e*Way and the external application. The **MUXPooler** is opened with the configured number of connections regardless of the number of connected applications. These connections are maintained by the e*Way to improve performance (connection/disconnection overhead is removed). The applications connected to the **MUXPooler** share these connections. If all of the connections are occupied, when another application tries to connect to the **MUXPooler**, a "Waiting for a free MUX" or "No MUX Available" message is produced.

4.1.1 Constructors

There are three constructors associated with the Muxpooler class that are used to instantiate an object:

- `public MUXPooler()`: Instantiates the object only. Each of the additional attributes must be set through a separate call individually.
  - **setHost** on page 22
  - **setPort** on page 23
  - **setSecondsToExpire** on page 23
  - **setTimeout** on page 24
- `public MUXPooler(String host, int port, int connectionCount,int timeout, int secondsToExpire)`: Instantiates the object and sets the values of the specified attributes.
- `public MUXPooler(String host, int port, int connectionCount,int timeout, int secondsToExpire, boolean debug)`: Instantiates the object and sets the values of the specified attributes. Included in these attributes is a flag to print debugging code to **System.out**. By default, it is not enabled as the other constructors set this to "False".

4.1.2 Methods

The Muxpooler class creates a user-defined number of MUX (multiplexer) connections to e*Gate and send/receive Events to e*Gate. The following public methods are included:

connect

Syntax

```
boolean connect()
```

Description

connect opens a connection to the Participating Host that is running the MUX e*Way.

Parameters

None.

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

disconnect

Syntax

```
boolean disconnect()
```

Description

disconnect closes the connection to the Participating Host that is running the MUX e*Way and waits for each connection to finish the associated transaction.

Parameters

None.

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

disconnect

Syntax

```
boolean disconnect(WaitOnMux)
```

Description

disconnect disconnects all connections to the MUX e*Way.

Parameters

Name	Type	Description
WaitOnMux	Boolean	Determines whether to wait for current transactions to complete before disconnecting.

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

getConnectionCount

Syntax

```
int getConnectionCount()
```

Description

getConnectionCount returns the number of MUX connections currently available.

Parameters

None.

Return Values

integer
Returns the total number of connections available within the **`MUXPooler`**. This includes free, non-used connections as well as the occupied connections.

getHost

Syntax

```
string getHost()
```

Description

getHost returns the host name.

Parameters

None.

Return Values

string
Returns the host name.

getPort

Syntax

```
int getPort()
```

Description

getPort returns the port number of the host machine.

Parameters

None.

Return Values

integer
Returns the port number.

getSecondsToExpire

Syntax

```
int getSecondsToExpire()
```

Description

getSecondsToExpire returns the expiration time in milliseconds.

Parameters

None.

Return Values

integer
Returns the number of milliseconds.

getTimeout

Syntax

```
int getTimeout()
```

Description

getTimeout returns the number of milliseconds to wait for a response before timeout.

Parameters

None.

Return Values

integer
Returns the number of milliseconds.

resizeMUXPool

Syntax

```
boolean resizeMUXPool(newSize)
```

Description

resizeMUXPool resizes the muxPool to the specified size.

Parameters

Name	Type	Description
newSize	integer	The number of connections of the muxPool

Return Values

Boolean
Returns true if the command executed successfully; otherwise, returns false.

Additional Information

resizeMUXPool is used to change the pool size at runtime (as necessary).

sendBytes

Syntax

```
byte[] sendBytes(bytes_array)
```

Description

sendBytes takes the message (Event) that is to be delivered into e*Gate, and returns e*Gate's response. A null string is returned if there is no response.

Parameters

Name	Type	Description
bytes_array	byte array	The message (Event) as a byte array

Return Values

byte array
Returns a byte array containing e*Gate's response if available; otherwise, returns null.

sendMessage

Syntax

```
string sendMessage(message)
```

Description

sendMessage takes the message (Event) that is to be delivered into e*Gate, and returns e*Gate's response. A null string is returned if there is no response.

Parameters

Name	Type	Description
message	string	The message (Event) to send

Return Values

string

Returns string containing e*Gate's response; otherwise, returns null if there was no response.

setConnectionCount

Syntax

```
void setConnectionCount(count)
```

Description

setConnectionCount sets the number of MUX connections allowed.

Parameters

Name	Type	Description
count	integer	The number of MUX connections

Return Values

void

Additional Information

setConnectionCount is used to initialize the Class.

setHost

Syntax

```
void setHost(host)
```

Description

setHost specifies the host name with which to establish a connection.

Parameters

Name	Type	Description
host	string	The host name.

Return Values

void

setPort

Syntax

```
void setPort(integer)
```

Description

setPort specifies the port number with which to establish a connection.

Parameters

Name	Type	Description
port	integer	The port name.

Return Values

void

setSecondsToExpire

Syntax

```
void setSecondsToExpire(seconds)
```

Description

setSecondsToExpire sets the expiration time to the specified value.

Parameters

Name	Type	Description
seconds	integer	The number of seconds

Return Values

void

setTimeout

Syntax

```
void setTimeout(timeout)
```

Description

setTimeout sets the timeout to the specified value.

Parameters

Name	Type	Description
timeout	integer	The number of seconds

Return Values

void

Chapter 5

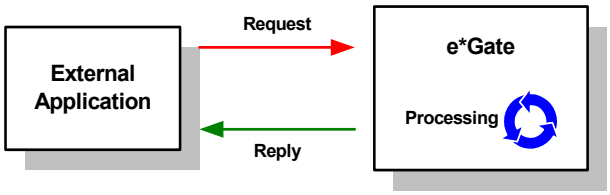
Implementation

5.1 The Request/Reply Concept

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

1. The Web server submits data (a **request)** to the e*Gate system.
2. The e*Gate system processes the data as required, communicating with other external ("backend") systems as necessary.
3. The e*Gate system returns data (a **response)** to the same thread within the Web server that submitted the request.

Figure 2 The Request/Reply concept

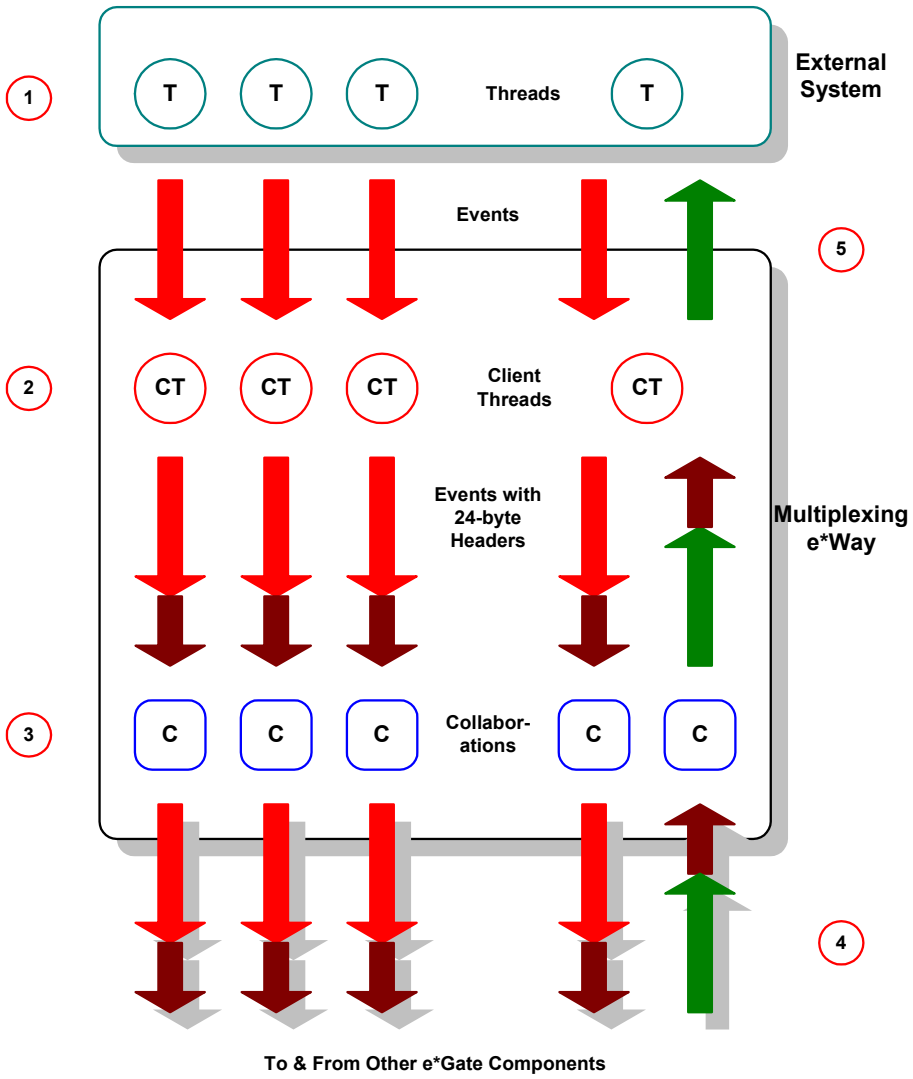


5.1.1 Request/Reply and the iPlanet e*Way Participating Host Components

The iPlanet e*Way Participating Host component is a multiplexing 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 3 illustrates how the multiplexing e*Way receives data from an external application and returns processed data to the same application.

Figure 3 Data flow through the multiplexing e*Way



1 The iPlanet App Server uses Java Servlets and Java Server Pages (JSP) to send the data to the multiplexing e*Way using a SeeBeyond-proprietary IP-based protocol.

2 Client threads within the e*Way package the data as e*Gate Events, adding a 24-byte header. Among other functions, this header provides "return address" information that can optionally be used to return data to the client thread that originated it.

   Each e*Way can handle up to 1,000 client threads at once. If your requirements demand more processing power, you can define more iPlanet e*Ways.

3 Collaborations within the e*Way perform any appropriate processing that may be required, and route the processed Events to other destinations (such as an external system for additional data retrieval or processing).

Note: *The 24-byte header **must** be preserved as the Events are processed through the e*Gate system.*

4 Processed data, still containing the original 24-byte header, is returned to the iPlanet e*Way.

5 The e*Way uses the 24-byte "return address" to identify the destination of the data to be returned to the external system.

6 The e*Way returns the data, minus the 24-byte header, to the client thread within the Web server.

5.1.2 The Request/Reply Schema

Request/Reply schemas have two classes of components:

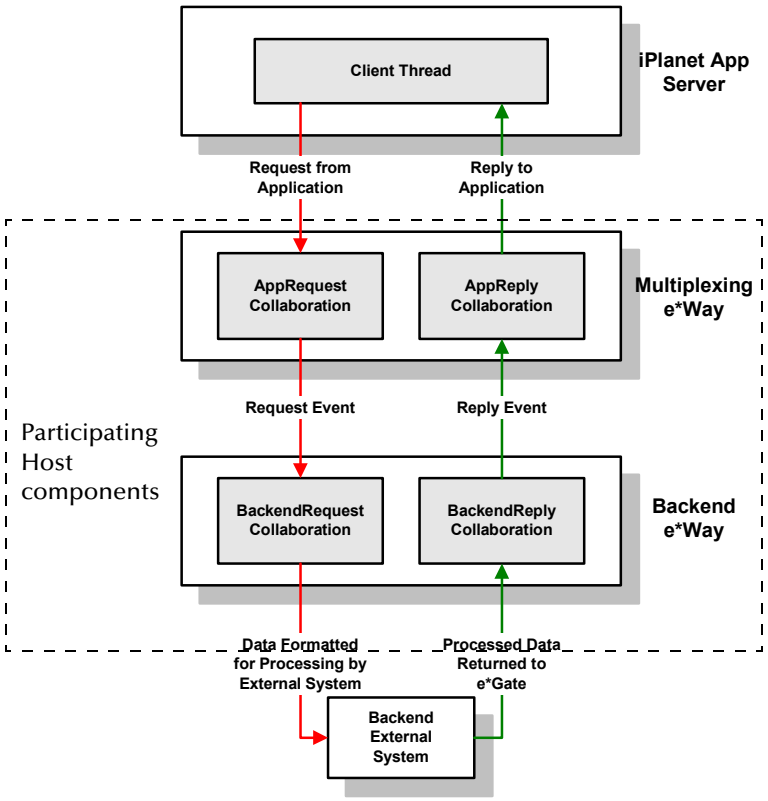
1 “Front-end” components that handle communications with the Web server. These components receive requests and route replies to the correct destination.

2 “Back-end” components that process the requests and compose the replies. These components also provide the bridge between the e*Gate system and your existing systems.

The iPlanet e*Way and its related Collaborations comprise the front-end components. A second e*Way and its related Collaborations comprise the back-end components (more e*Ways may be added to communicate with more external systems as required). The back-end e*Way(s) can be of any type required to communicate with the external system(s).

Figure 4 below illustrates a typical Request/Reply schema.

Figure 4 The Request/Reply schema



Implementation using Servlets

1. The user submits data to the Web server via a browser form.
2. The Web server sends the request information to the servlet.
3. The servlet packages the request and invokes the e*Gate extension subroutines, and forwards the data to the Participating Host.
4. Data enters the Participating Host through the multiplexing e*Way's WebRequest Collaboration.
5. The WebRequest Collaboration publishes the Request Event.
6. The BackendRequest Collaboration within the back-end e*Way subscribes to the Request Event, and processes the data as appropriate, and routes it to the external backend system.
7. The external backend system processes the data and returns it to e*Gate via the Backend e*Way.
8. The Backend e*Way's BackendReply Collaboration publishes the data as the Reply Event.
9. The WebReply Collaboration within the multiplexing e*Way subscribes to the Reply Event.

10 The multiplexing e*Way returns the processed data to the servlet.

11 The servlet builds a response and passes it to the server.

12 The Web server sends the response back to the client.

Implementation using JSP Pages

1 The user sends a request to the Web browser.

2 The Web browser makes a request to the JSP page.

3 The JSP page parses the contents of the JSP page and generates a temporary servlet.

4 The servlet packages the request and invokes the e*Gate extension subroutines, and forwards the data to the Participating Host.

5 Data enters the Participating Host through the multiplexing e*Way's WebRequest Collaboration.

6 The WebRequest Collaboration publishes the Request Event.

7 The BackendRequest Collaboration within the back-end e*Way subscribes to the Request Event, and processes the data as appropriate, and routes it to the external backend system.

8 The external backend system processes the data and returns it to e*Gate via the Backend e*Way.

9 The Backend e*Way's BackendReply Collaboration publishes the data as the Reply Event.

10 The WebReply Collaboration within the multiplexing e*Way subscribes to the Reply Event.

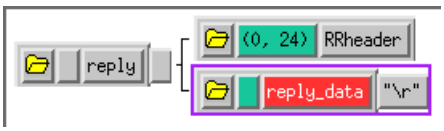
11 The multiplexing e*Way returns the processed data to the servlet. The combination of static HTML and graphics combined with the dynamic elements specified in the original JSP page definition are sent to the Web browser through the output stream of the servlet's response object.

5.1.3 ETDs and Form Data

As discussed in **"Request/Reply and the iPlanet e*Way Participating Host Components" on page 25**, the iPlanet e*Way maintains "return address" information in a 24-byte header that must be preserved as the data flows through the e*Gate system.

The simplest Event Type Definition (ETD) that can be used within a Request/Reply schema has two nodes: one for the header, the second for the remainder of the Event data.

Figure 5 The simplest Request/Reply ETD



This ETD is sufficient if you wish to send data through the e*Gate system simply as a blob. For example, you can compose the reply to the Web server as a block of completely formatted HTML code, then return that reply using the above ETD.

Although the simple ETD in Figure 5 can be sufficient for reply data, request (input) data will probably have a more complex structure. To accommodate this, you must add the additional structure to the basic Request/Reply ETD:

1. Be sure to maintain the 24-byte "header" node unchanged.
2. Add one subnode beneath the "data" node for each element of input data.
3. Modify those subnodes as necessary (for example, use appropriate delimiters or record lengths).

Figure 6 below illustrates an ETD that describes delimited data (for example, as in the data "First name^Last name").

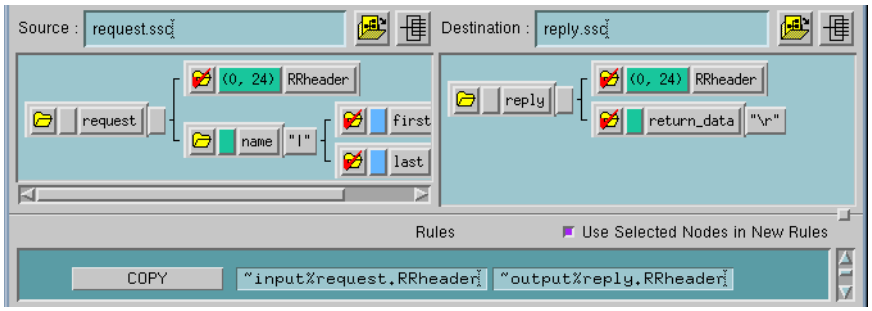
Figure 6 A Request/Reply ETD for delimited data

Of course, the more complex the form that collects user data, the more complex the ETD that describes the form data must become. Be sure that the Request ETD meets the requirements for input to the backend system.

Collaboration Rules and the iPlanet Header

Collaboration Rules that manipulate data between ETDs must preserve the Request/Reply header (in the figures above, "RRheader"). Be sure that each Collaboration Rule that manipulates Request/Reply data copies the contents of the Request/Reply header from the source ETD to the target ETD (as shown in Figure 7 below).

Figure 7 Copying the Request/Reply header



5.2 Using the Java Servlet and JSP Page

The minimal implementation of the iPlanet e*Way via a iPlanet App Server would work as follows:

1. A Java servlet or JSP page for accepting requests from the Web server or Web browser.
2. The servlet sends the user request to the e*Way and posts the result to the Web server. The JSP page sends the user request to the e*Way and posts the result to the Web browser.

The Java Servlet is a set of Java classes which defines a standard interface between a Web client and a Web server. Client requests are made to the Web server, which then invokes the servlet to service the request through this interface. The Java servlet API is a standard Java Extension API.

For the sample provided, the API is composed of two packages:

- javax.servlet
- javax.servlet.http

The 'Servlet' interface class defines the methods which servlets must implement, including a 'Service0' method for the handling of requests.

'HttpServlets' provide additional methods for the processing of HTTP requests such as GET (doGet method) and POST (doPost method).

Java Server pages are composed of standard HTML tags and JSP tags. The available JSP tags defined in the JSP 1.0 specification are categorized as the following:

- directives
- declarations
- scriplets
- comments
- expressions

Please refer to the sample files for additional information and the iPlanet App Server for detailed instructions on how to create a Web application on the Administrative Console.

Additional information can be found in commented sample files (see the next section for more information).

5.3 Sample Implementation

A sample schema for setting up the iPlanet e*Way server is provided on the e*Gate installation CD-ROM in the directory

/samples/ewiplanet

1 Copy all the files from the e*Gate installation CD-ROM samples directory **/samples/ewiplanet** to a temporary working directory on the system on which the iPlanet App Server is installed.

2 Copy the sample file **SampleServlet.class** to the iPlanet directory where your Web application is located. For example:

/ServletDirectory

Note: *A user cannot run **stcregutil.exe** on machines that do not have the e*Gate Participating Host installed.*

3 Use **stcregutil.exe** to import the RequestReply schema:

```
stcregutil.exe -rh localhost -rs RequestReply
     -un username -up passwd -i RequestReply.exp
```

4 Again, use **stcregutil.exe** to import the configuration files:

```
stcregutil.exe -rh localhost -rs RequestReply
     -un username -up passwd -fc . -ctl RequestReply.ctl
```

5 Start the Control Broker for this new schema:

```
stccb.exe -ln RequestReply_cb -rh localhost -rs RequestReply
     -un username -up passwd -sm
```

These instructions also appear in "readme" files in the iPlanet e*Way samples directory (**/samples/ewiplanet**).

Once the client and server are set up, you can test the sample files using a Web browser. Open the browser and enter the following URLs:

http://localhost/webapp/your_application_name/SampleServlet

5.4 SampleServlet.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.stc.ewip.*;

public class SampleServlet extends javax.servlet.http.HttpServlet
{
      int port;
      String host;
      MUXPooler muxPool;

      public SampleServlet()
      {
            super();
            host = "localhost";
            port = 26051;
      }

      public void init(ServletConfig config) throws ServletException
      {
            super.init(config);
            //
            // This will instantiate our MUX Pool and make it available to all hits to this servlet
            //
            muxPool = new MUXPooler(host, port, 20, 10000, 10, false);
      }
```

```
      public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
      {
            doPost(request, response);
      }

      public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
      {
            boolean fConnected;
            String inputString;
            String url;
            String portString;
            String muxReturnString;
            String reconnect;
            PrintWriter out;

            fConnected = true;
            url = HttpUtils.getRequestURL(request).toString();
            response.setContentType("text/html");
            out = response.getWriter();
            out.println("<HTML><TITLE>e*Gate Servlet Sample</TITLE><BODY>");
            out.println("<H2>e*Gate Servlet Example</H2>");
            out.println("<HR>");

            //
            // extract our form data
            //
            inputString = request.getParameter("muxInput");
            reconnect = request.getParameter("reconnect");

            // ************************************************************
            // ************************************************************
            //
            // THIS IS THE ONLY PART WHERE WE USE THE MUX BESIDED IN init()
            // check for an input message and send to e*Gate
            //
            if (reconnect != null && reconnect.equals("YES"))
            {
                  //
                  // extract the host and port from our form
                  //
                  host = request.getParameter("host");
                  portString = request.getParameter("port");
                  try
                  {
                        port = Integer.parseInt(portString);
                  }
                  catch(NumberFormatException e)
                  {}

                  if (host == null || host.equals(""))
                        host = "localhost";
                  if (port == 0)
                        port = 26051;

                  //
                  // now connect to the MUX
                  //
                  fConnected = false;
                  muxPool.setHost(host);
                  muxPool.setPort(port);
                  fConnected = muxPool.connect();
            }

            if (fConnected)
            {
                  if (inputString != null && inputString.equals("") == false)
                  {
                        muxReturnString = muxPool.sendMessage(inputString);
                        if (muxReturnString == null)
                              muxReturnString = "ERROR! - mux returned null";

                        out.println("<H3>e*Gate returned: </H3><PRE>" + muxReturnString + "</
PRE>");
                        out.println("<HR>");
                  }
            }
            else
            {
                  muxReturnString = "ERROR! - muxpooler.connect() failed";
                  out.println("<H3>e*Gate returned: </H3><PRE>" + muxReturnString + "\nhost: " +
host + "  port: " + port + "</PRE>");
                  out.println("<HR>");
            }
            //
            // ************************************************************
            // ************************************************************


            //
            // now create our form for a sample input message
```

```
                //
                out.println("<FORM ACTION=" + url + " METHOD=GET>");
                out.println("<TABLE ALIGN=\"LEFT\"><TR><TD>");
                out.println("<B>e*Gate test message:  </B><BR>");

                out.println("<TEXTAREA NAME=\"muxInput\" COLS=\"32\" ROWS=\"5\" WRAP=PHYSICAL></
TEXTAREA>" );
                out.println("<BR><BR><BR>");
                out.println("<B>mux host </B>");
                out.println("<INPUT NAME=\"host\" SIZE=\"30\" value=\"" + host + "\">" );
                out.println("<BR><B>mux port </B>");
                out.println("<INPUT NAME=\"port\" SIZE=\"7\" value=\"" + port + "\">" );
                out.println("  <B>reconnect </B> <SELECT
NAME=\"reconnect\"><OPTION>NO<OPTION>YES</SELECT>");

                out.println("<BR>");
                out.println("</TD></TR>");
                out.println("<TR><TD ALIGN=\"RIGHT\">");
                out.println("<BR><INPUT TYPE=\"submit\" VALUE=\"Submit\">");
                out.println("</TD></TR></TABLE>");
                out.println("<BR><BR></FORM>");
                out.println("</BODY></HTML>");
                out.close();
        }
}
```

Index