SeeBeyond ICAN Suite

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

Release 5.0.5 for Schema Run-time Environment (SRE)

Monk Version

Version 20050406092535.

Contents

Chapter 1

Introduction 7

HTTP(S) e*Way: Overview 7
- Using Clear HTTP 8
- Intended Reader 8
- Components 8
- Basic Information 9

General Operation 9

Supported Operating Systems 10

System Requirements 11

Chapter 2

Installation 12

Windows Systems 12
- Pre-installation 12
- Installation Procedure 12

UNIX Systems 13
- Pre-installation 13
- Installation Procedure 13

Files/Directories Created by the Installation 14

Chapter 3

Clear HTTP Implementation 16

e*Way Implementation/Clear HTTP: Overview 16

Sample Configurations 17
- Creating a Schema Using httpnossl-outgoing 17
- Creating a Schema Using httpnossl-exchange 22

Sample Monk Scripts 25
- GET (Inbound) Example (HTTP_get) 25
- POST (Outbound) Example (HTTP_post) 26
- Input File based Example (AUTO_HTTP) 26

Chapter 4

Clear HTTP Functions 28

HTTP Functions: Introduction 28

Basic Functions 28

HTTP Standard Functions 29

HTTP Monk Functions 35

Chapter 5

Secure Sockets Layer Operation 49

Using Secure Sockets Layer: Overview 49

Certificates and Security 49

Using the openssl Utility 50

Working with PKCS12 files 50

Converting PKCS12 Files to PEM Files 50

Converting DER Files to PEM Files 51

Converting Other Formats 51

SSL Handshaking 51

Chapter 6

HTTP(S) e*Way Configuration 55

Introduction 55

e*Way Configuration Parameters 55

General Settings 56

Journal File Name 56

Max Resends Per Message 56

Max Failed Messages 56

Forward External Errors 57

Communication Setup 57

Exchange Data Interval 57

Zero Wait Between Successful Exchanges 58

Start Exchange Data Schedule 58

Stop Exchange Data Schedule 59

Down Timeout 59

Up Timeout 59

Resend Timeout 59

Monk Configuration 60

e*Way Structure 60

Operational Details 61

How to Specify Function Names or File Names 67

Additional Path 67

Auxiliary Library Directories 68

Monk Environment Initialization File 68
Startup Function 69
Process Outgoing Message Function 69
Exchange Data with External Function 70
External Connection Establishment Function 71
External Connection Verification Function 72
External Connection Shutdown Function 72
Positive Acknowledgment Function 73
Negative Acknowledgment Function 73
Shutdown Command Notification Function 74
HTTP Configuration 74
Request 74
Timeout 75
URL 75
User Name 75
Encrypted Password 76
Agent 76
Content-type 76
Request-content 76
Accept-type 77
HTTP Proxy Configuration 77
Use Proxy Server 77
User Name 77
Encrypted Password 77
Server Address 78
Port Number 78
HTTP(S) Configuration 78
Trusted CA Certificates Directory 78
Use Client Certificate Map 78
Client Certificate Map File 79
Working with Certificates 79
Required Certificate Format 79
Obtaining Certificates 80
Obtaining CA Certificates From Secure Sites using Internet Explorer 80
Exporting CA Certificates 81
Working with Client Certificate/Key Pairs 82
Importing Certificates to the e*Gate Registry 83

Chapter 7

HTTP(S) e*Way Implementation 84

HTTP(S) e*Way Implementation: Overview 84

Creating Event Type Definitions from Form Data 85
Creating Event Type Definitions using Command-line Utilities 85
Creating Event Type Definitions from the ETD Editor 86

Sample Configurations 87
Creating a Schema Using http-outgoing 88
Creating a Schema Using http-exchange 93

Sample Monk Scripts 97

GET (Inbound) Example (HTTP_get) 98
POST (Outbound) Example (HTTP_post) 98
Sample Input Data (AUTO_HTTP) 99
GET (Inbound) Example (HTTPS_get) 100

Index 102

Chapter 1

Introduction

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

1.1 HTTP(S) e*Way: Overview

The HTTP(S) e*Way allows integration with third-party applications over the Internet using the hyper-text transfer protocol (HTTP) and HTTP with the Secure Sockets Layer (SSL) feature. This e*Way supports both the GET and POST methods. This version of the HTTP(S) e*Way is enabled by the Monk programming language.

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

GET and POST Methods

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

The following example shows a URL-encoded query string:

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

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

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

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

Taking the above URL as an example, the user specifies **http://google.yahoo.com/bin/query** as the URL, then specifies the name value pair separately. The HTTP client allows for specification of the URL and n-number of value pairs through its methods.

SSL Protocol

Beyond simple data transfer with the HTTP protocol, the HTTP(S) e*Way offers the option of using HTTP to provide secure data transport using the SSL protocol. This capability provides privacy and authentication by encrypting the data in transit between the e*Way and the server, and by verifying both the client's and server's identities before commencing the transaction. The e*Way can also set certificate and private key information that is required to communicate with some secure servers.

Note: *See* **Chapter 5** *for details on the e*Way's SSL feature.*

1.1.1 Using Clear HTTP

The HTTP(S) e*Way can be used without the SSL features. When you use clear HTTP, the e*Way is equally functional in an HTTP environment. For information on how to use the e*Way with clear HTTP (without SSL), see the following chapters:

- **Chapter 3**: **"Clear HTTP Implementation"**
- **Chapter 4**: **"Clear HTTP Functions"**

1.1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate Integrator system; to have high-level knowledge of Windows and/or UNIX operations and administration; to be thoroughly familiar with HTTP(S) certification, Web servers, and Windows-style graphical user interface (GUI) operations.

1.1.3 Components

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

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor GUI uses to define configuration parameters
- Function-library files, **stc_monkhttp.dll** and **stc_monkhttp_nossl.dll**
- HTML Converter, a tool that builds a Monk ETDs from a sample HTML page (see **"Creating Event Type Definitions from Form Data" on page 85**).

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

Note: *For complete information on the HTML Converter, see the* ***HTML Converter User's Guide****.*

1.1.4 Basic Information

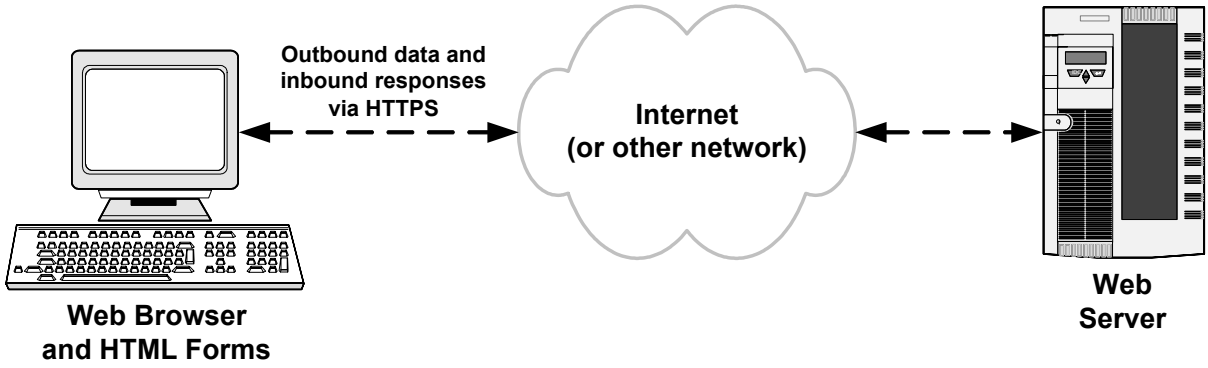
The following information applies to the HTTP(S) e*Way:

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

1.2 General Operation

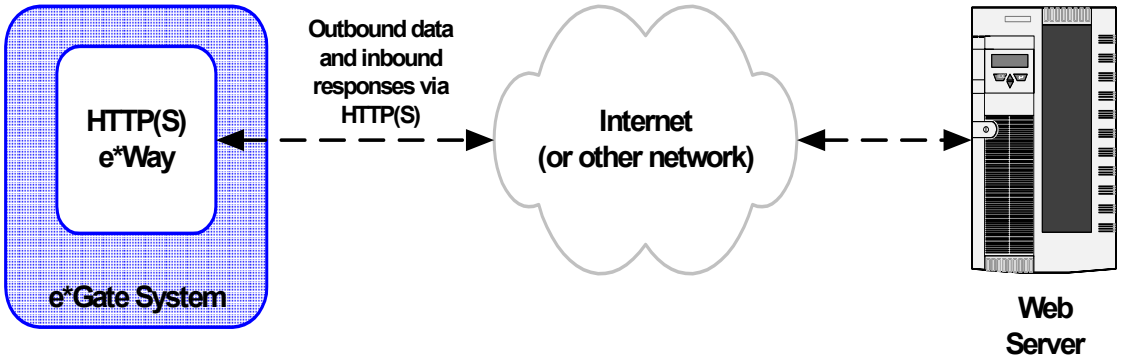
In a typical data exchange using HTTP(S), a user sends requests to a Web server using a Web browser (for instance, when sending an order to an online shopping service using HTML forms). See Figure 1 for details.

Figure 1 HTTP(S) Data Exchange Using Browser



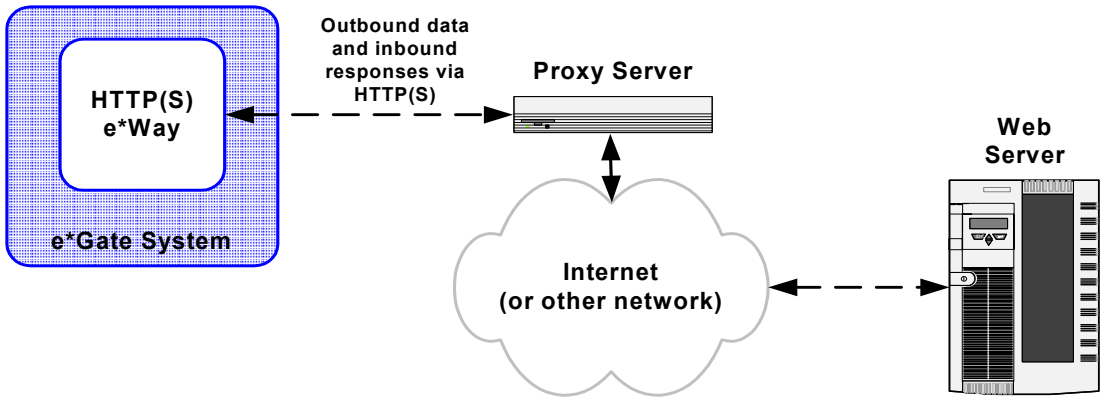
In an e*Gate implementation, the HTTP(S) e*Way exchanges data using the same HTTP(S) methods as a browser might (see Figure 2).

Figure 2 HTTP(S) Data Exchange Using the HTTP(S) e*Way



The HTTP(S) e*Way can also be configured to operate through a proxy server when the e*Gate components are separated from the target Web server by a fire wall (see Figure 3).

Figure 3 HTTP(S) Data Exchange Through a Fire Wall



1.3 Supported Operating Systems

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

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

1.4 System Requirements

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

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

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

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

External System Requirements

There are no external system requirements.

Chapter 2

Installation

This chapter explains how to install the HTTP(S) e*Way Intelligent Adapter.

2.1 Windows Systems

2.1.1 Pre-installation

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

2.1.2 Installation Procedure

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

1. Log in as an Administrator on the workstation where you want to install the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's "Autorun" feature is enabled, the setup application launches automatically; skip to the next step. Otherwise, use Windows Explorer to launch the file **setup.exe** on the CD-ROM drive.
4. The **InstallShield** setup application launches. Follow the on-screen instructions to install the e*Way. Select the **HTTP(S) e*Way** installation option.

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

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

5. After the installation is complete, exit the install utility and launch the e*Gate Integrator Schema Designer graphical user interface (GUI).
6. In the Component editor, create a new e*Way.
7. Display the new e*Way properties.
8. On the **General** tab, under **Executable File**, click **Find**.

9 Select the file **stcewgenericmonk.exe**.

10 Under **Configuration file**, click **New**.

11 From the **Select an e*Way template** list, select **stcewhttp** and click **OK**.

12 The e*Way Editor GUI launches. Make any necessary changes, then save the configuration file.

13 You return to the e*Way's properties sheet. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 6**.

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, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.*

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

2.2 UNIX Systems

2.2.1 Pre-installation

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

2.2.2 Installation Procedure

To install the HTTP(S) e*Way on a UNIX system

1 Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.

2 If necessary, mount the CD-ROM drive.

3 At the shell prompt, type

cd /cdrom/setup

4 Start the installation script by typing:

setup.sh

5 A menu of options will appear. Select the **install e*Way** option. Then, follow any additional on-screen directions.

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

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

6 After installation is complete, exit the installation utility and launch the e*Gate Schema Designer.

7 In the Component editor, create a new e*Way.

8 Display the new e*Way's properties.

9 On the General tab, under **Executable File**, click **Find**.

10 Select the file **stcewgenericmonk.exe**.

11 Under **Configuration file**, click **New**.

12 From the **Select an e*Way template** list, select **stcewhttp** and click **OK**.

13 The e*Way Editor GUI launches. Make any necessary changes, then save the configuration file.

14 You return to the e*Way's properties sheet. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 6**.

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, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions.*

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

2.3 Files/Directories Created by the Installation

The HTTP(S) e*Way installation process installs the files shown in Table 1 within the e*Gate directory tree. Files are installed within the **eGate\client** tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files Created by HTTP(S) e*Way Installation

e*Gate Directory	File(s)
bin\	stcewgenericmonk.exe
bin\	stc_monkhttp.dll
bin\	stcewgenericmonk.exe stc_monkfilesys.dll stc_monkhttp_nossl.dll
configs\stcewgenericmonk\	stcewhttp.def
configs\stcewgenericmonk\	stcewhttpnossl.def

Table 1 Files Created by HTTP(S) e*Way Installation (Continued)

e*Gate Directory	File(s)
monk_library	httpnossl.gui
monk_library\	http.gui
monk_library\ewhttp\	http-ack.monk http-nack.monk http-connect.monk http-exchange.monk http-init.monk http-notify.monk http-outgoing.monk http-shutdown.monk http-startup.monk http-verify.monk
monk_library\ewhttpnossl\	httpnossl-ack.monk httpnossl-nack.monk httpnossl-connect.monk httpnossl-exchange.monk httpnossl-init.monk httpnossl-notify.monk httpnossl-outgoing.monk httpnossl-shutdown.monk httpnossl-startup.monk httpnossl-verify.monk
pkicerts\client\	certmap.txt
pkicerts\trustedcas\	GTECyberTrustGlobalRoot.cer MicrosoftRootAuthority.cer SecureServerCertificationAuthority.cer ThawtePremiumServerCA.cer ThawteServerCA.cer verisign_class3.cer

Chapter 3

Clear HTTP Implementation

This chapter explains how to implement the HTTP(S) e*Way Intelligent Adapter with clear hyper-text transfer protocol (HTTP), in a production environment.

Note: *This operation does **not** provide the Secure Sockets Layer (SSL) feature. If you want to implement the HTTP(S) e*Way with this feature, see* **Chapter 7**.

3.1 e*Way Implementation/Clear HTTP: Overview

To implement the HTTP(S) e*Way with clear HTTP (without SSL) within the e*Gate Integrator system, you must do the following operations:

- Define Event Type Definitions (ETDs) to package the data being exchanged with the external system.

Note: *The HTTP(S) e*Way Extension (**stc_monkhttpnossl.dll**) is not thread-safe. It must only be used in an e*Way or a **single Collaboration** in a Business Object Broker (BOB).*

- In the e*Gate Schema Designer graphical user interface (GUI), do the following steps:
  - Define Collaboration Rules to process Event data.
  - Define any Intelligent Queues (IQs) to which Event data is published before sending it to the external system.
  - Define the e*Way component.
  - Within the e*Way component, configure Collaborations to apply the required Collaboration Rules.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's online Help or the **e*Gate Integrator User's Guide**.*

- Use the e*Way Editor to set the e*Way's configuration parameters (this procedure is explained in **Chapter 6**).
- Be sure that any other e*Gate components are configured as necessary to complete the schema.
- Test the schema and make any necessary corrections.

See **"Sample Monk Scripts" on page 25** for examples of how the previous steps are combined to create a working implementation.

Note: *The delimiters for the configuration file must not appear within the URL string. The default delimiter set contains the equals sign (=), to modify this delimiter, open the configuration file, select* ***Options***, ***Config Delimiters****, on the task bar, modify the value of delimiter 3 with a value that will not conflict with the search string.*

Creating ETDs from Form Data

Creating ETDs for the HTTP(S) e*Way (no SSL) uses the same procedures as those used to create ETDs for the HTTP(S) e*Way (with SSL). See **"Creating Event Type Definitions from Form Data" on page 85** for details.

3.2 Sample Configurations

This section describes several sample implementations for the HTTP(S) e*Way.

3.2.1 Creating a Schema Using httpnossl-outgoing

This section demonstrates how to set up a basic schema using the **httpnossl-outgoing** function. In this sample, data is drawn from a text file using the file e*Way and sent to an external system using the HTTP(S) e*Way. The data returned from the external system is received by the HTTP(S) e*Way, then forwarded to another file e*Way and stored in an output file on the local system (see **Figure 18 on page 88**).

This schema requires a number of components as illustrated in **Figure 19 on page 89**.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's online Help or the* ***e*Gate Integrator User's Guide****.*

To create a schema using httpnossl-outgoing

1. Log into the e*Gate Schema Designer and click **New** to create a new schema. Name the schema "http_sample_1."

   The Schema Designer main screen appears.
2. If the Navigator's **Components** tab is not selected already, select it now.
3. Create an Event Type named "In."

4 Display the properties of the **In** Event Type. Then, use the **Find** button, navigate to the **common** folder to assign the file **GenericInEvent.ssc**.

5 Create a Collaboration Rule named "Passthrough_Data."

6 Edit the Properties of this Collaboration Rule as follows:

Service	Pass Through
Subscription	In (the Event Type defined in Step 1 above)
Publication	In (Event Type defined in Step 1 above)

7 Create two IQs, named "Inbound_IQ" and "HTTP_IQ."

8 Create an e*Way named "Inbound."

9 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.

The next part of the procedure requires that you launch the e*Way editor and define the file-based e*Way's properties.

1 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

2 Using the e*Way Editor, do the following configuration settings:

Section	Parameter and setting
General Settings	AllowIncoming: Yes AllowOutgoing: No
Poller (inbound) Settings	Polldirectory: C:\TEMP (or other "temporary" directory) Input File Mask: leave unchanged

3 Save the settings, promote to run time, and exit the e*Way Editor.

4 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create a Collaboration for the Inbound e*Way as follows:

1 Open the **Inbound** e*Way and create a Collaboration named "Inbound_collab."

2 Set the Collaboration's properties as follows:

Collaboration Rule	Passthrough_Data
Subscriptions	Event: In Source: <External>.
Publications	Event: In Publish to: Inbound_IQ.

Now that the inbound e*Way is completely configured, you must create an outbound HTTP(S) e*Way.

1 Create a new e*Way component named "http_eway."

2 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewgenericmonk.exe**.

3 Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewhttpnossl**.

4 Use the e*Way Editor to define the following parameters:

Section	Parameter and Settings
General Settings	Leave all settings unchanged
Communication Setup	Exchange Data Interval: 0 (zero) Zero Wait Between Successful Exchanges: No
Monk Configuration	Auxiliary Library Directories: monk_library/ewhttp Monk Environment Initialization File: monk_library/ewhttpnossl/httpnossl-init.monk Startup Function: httpnossl-startup Process Outgoing Message Function: httpnossl-outgoing Exchange Data With External Function: httpnossl-exchange External Connection Establishment Function: httpnossl-connect External Connection Verification Function: httpnossl-verify External Connection Shutdown Function: httpnossl-shutdown Positive Acknowledgment Function: httpnossl-ack Negative Acknowledgment Function: httpnossl-nack The remaining parameters may be left blank for this sample.
HTTP Configuration	Timeout: 5000 User Name: enter an appropriate user name if necessary Encrypted Password: enter an appropriate password if necessary Agent: e*Gate HTTP(S) e*Way Content-type: Content-Type:application/x-www-form-urlencoded Accept-type: accept:text/* The remaining parameters may use the default values.
HTTP Proxy Configuration	Leave blank unless required
HTTP Configuration	Leave blank to test basic HTTP functionality; if required, enter any necessary information to test HTTP functionality

5 Save the settings, promote to run time, and exit the e*Way Editor.

6 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's Main window.

Important: *The above code loads the certificate (s) and private key (s) from the specified directory.*

Next, create the Collaboration for the HTTP(S) e*Way.

1. Select the **http_eway** component and create a Collaboration named "http_collab1."
2. Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: Inbound_collab
Publications	Event: In Publish to: <External>

3. Create a second Collaboration for the **http_eway**, naming it "http_collab2."
4. Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: <External>
Publications	Event: In Publish to: HTTP_IQ

Now create and configure the final e*Way component.

1. Create a new e*Way named "Outbound."
2. In its Properties Page, specify the executable file of "Outbound" as **stcewfile.exe**.
3. Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.
4. With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.
5. Using the e*Way Editor, configure the following settings:

Section	Parameter and setting
General Settings	AllowIncoming: No AllowOutgoing: Yes
Outbound (sender) Settings	Output directory: C:\TEMP (or other "temporary" directory) Output File Name: httpnossl_out.txt

6. Save the settings, promote to run time, and exit the e*Way Editor.
7. When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.
8. Create a Collaboration for the "Outbound" e*Way, naming it "outbound_collab."

9 Set the Collaboration's properties as follows:

Collaboration Rules:	Passthrough_Data
Subscriptions	Event: In Source: http_collab2
Publications	Event: In Publish to: <External>

The Schema Designer configuration is now complete. Now, you must create some test data which will be sent via HTTP to external Web sites. The results of these requests will be saved to the output data file.

Note: *The sites recommended within the test data are publicly available sites, and the test data was accurate at the time this guide was published. If any of the recommended sites are no longer available, or you wish to replace them with your own test sites, please make the appropriate substitutions.*

1 Use a text editor to create an input file. Create an Input File, using any ASCII text editor. The input must have the following format (the pipe symbol "|" delimits each field):

```
URL|POST or GET|data (POST only)
```

The following sample can also be used as your test data, changing "somesite" to a valid HTTP site name:

```
http://info.somesite.com|GET|
http://finance.somesite.asp|POST|s=amd&d=v1
http://search.somesite.com/cgi-bin/
search|POST|search=Mars+missions
http://finance.somesite.com/q|GET|s=amd&d=v1
http://finance.somesite.com/q|GET|s=amd+&d=v4
http://finance.somesite.com/q|GET|s=amd&d=v1
```

Note: *When using an input file, it is necessary to modify the fields within the configuration file to match those within the input file, or to leave the fields blank. If a field in the configuration file, such as the* ***Request-content*** *parameter contains a string, and it does not appear within the input file, e*Gate will attempt to append the information. If within the input file, the delimiters are left empty the action within the configuration file will be used.*

2 Save the file as **c:\temp\testdata.fin** (if you specified a different input directory, please make the appropriate substitution).

Launch the sample schema. If the schema was configured properly and your connection to the test sites is good, you should find response data from your requests in the file **C:\TEMP\httpnossl_out.txt** (if you specified a different output directory, please make the appropriate substitution).

3.2.2 Creating a Schema Using httpnossl-exchange

This schema, which illustrates the use of the Monk function **httpnossl-exchange**, is simpler than the one illustrated in **"Creating a Schema Using httpnossl-outgoing" on page 17**. Rather than using an inbound e*Way, the data to be sent to the external Web server is hard-coded into the HTTP(S) e*Way's configuration using the e*Way editor. Except for this change, the architecture is the same.

See **Figure 20 on page 94** for a diagram of the schema.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's online Help of the* ***e*Gate Integrator User's Guide****.*

To create a schema using httpnossl-exchange

1. Log into the e*Gate Schema Designer and select the New to create a new schema.
2. Enter the new schema name.
3. Create an Event Type named "In."
4. Display the properties of the **In** Event Type. Then, use the **Find** button to assign the file **GenericInEvent.ssc**.
5. Create a Collaboration Rule named "Passthrough_Data."
6. Edit the Properties of this Collaboration Rule as follows:

Service	Pass Through
Subscription	In (the Event Type defined in Step 1 above)
Publication	In (Event Type defined in Step 1 above)

7. Create an Intelligent Queue, named "HTTP_IQ."

You must create an outbound HTTP(S) e*Way.

1. Create a new e*Way component named "http_eway."
2. Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewgenericmonk.exe**.
3. Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewhttpnossl**.
4. Use the e*Way Editor to define the following parameters:

Section	Parameter and Settings
General Settings	Leave all settings unchanged
Communication Setup	Exchange Data Interval: 10 (ten) Zero Wait Between Successful Exchanges: No

Section	Parameter and Settings (Continued)
Monk Configuration	Auxiliary Library Directories: monk_library/ewhttpnossl Monk Environment Initialization File: monk_library/ewhttpnossl:/httpnossl-init.monk Startup Function: httpnossl-startup Process Outgoing Message Function: httpnossl-outgoing Exchange Data With External Function: httpnossl-exchange External Connection Establishment Function: httpnossl-connect External Connection Verification Function: httpnossl-verify External Connection Shutdown Function: httpnossl-shutdown Positive Acknowledgment Function: httpnossl-ack Negative Acknowledgment Function: httpnossl-nack The remaining parameters may be left blank for this sample.
HTTP Configuration	Request: GET Timeout: 5000 URL: enter an appropriate URL to contact. User Name: enter an appropriate user name if necessary Encrypted Password: enter an appropriate password if necessary Agent: e*Gate HTTP(S) e*Way Content-type: Content-Type:application/x-www-form-urlencoded Request-content: Leave this entry blank (because this is a sample using GET; fill in this field when using the POST method). Accept-type: accept:text/* The remaining parameters may use the default values.
HTTP Proxy Configuration	Leave blank unless required

5 Save the settings, promote to run time, and exit the e*Way Editor.

6 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

In the next step, you modify the initialization function (**httpnossl-init**) loads the correct **.dll**.

1 From the Schema Designer's **File** menu, select **Edit File**.

2 Open the file **monk_library\ewhttpnossl\httpnossl-init.monk**.

3 Verify that the **stc_monkhttp_nossl.dll** is the specified file in the **(load-extension)** function call.

Save and exit the editor of the text file. Verify that the files are in the appropriate location.

Next, create the Collaboration for the HTTP(S) e*Way.

1 Create a Collaboration for the **http_eway**, naming it "http_collab2."

2 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: <External>
Publications	Event: In Publish to: HTTP_IQ

Now create and configure the final e*Way component as follows:

1 Create a new e*Way named "Outbound."

2 In its Properties Page, specify the executable file of "Outbound" as **stcewfile.exe**.

3 Display the e*Way's properties. Then, use the **Find** button, navigate to the "**bin**" folder to assign the file **stcewfile.exe**.

4 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

5 Using the e*Way Editor, configuration the following settings:

Section	Parameter and setting
General Settings	AllowIncoming: No AllowOutgoing: Yes
Outbound (sender) Settings	Output directory: C:\TEMP (or other "temporary" directory) Output File Name: httpnossl_out.txt

6 Save the settings, promote to run time, and exit the e*Way Editor.

7 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

8 Create a Collaboration for the "Outbound" e*Way, naming it "outbound_collab."

9 Set the Collaboration's properties as follows:

Collaboration Rules:	Passthrough_Data
Subscriptions	Event: In Source: http_collab2
Publications	Event: In Publish to: <External>

The Schema Designer configuration is now complete. Now, you must create some test data which will be sent via HTTP to external Web sites. The results of these requests are saved to the output data file.

Note: *The sites recommended within the test data are publicly available sites, and the test data was accurate at the time this guide was published. If any of the recommended sites are no longer available, or you wish to replace them with your own test sites, please make the appropriate substitutions.*

3.3 Sample Monk Scripts

This section describes several sample implementations for the HTTP(S) e*Way.

The samples in this section can be run using the **stctrans** command-line utility. They do not require a complete e*Gate schema configuration to function, and are designed to illustrate the principles involved in creating your own custom Monk scripts. The library (.**dll**) files to be loaded and the script to be tested must be in the load path (or, for simplicity's sake, may be placed in the connected directory).

Note: *See the **Monk Developer's Reference** for more information about the load path.*

The syntax of the **stctrans** utility is

```
stctrans monk_file.monk
```

Additional command-line flags are available; enter **stctrans -h** to display a list, or see the *e*Gate Integrator System Administration and Operations Guide* for more information.

The sample files may be created using any text editor. The samples use a generic "www.sitename.com" site name; before testing any script, replace the generic name with a working site name.

3.3.1 GET (Inbound) Example (HTTP_get)

The script in this section retrieves the URL **http://www.somesite.com** and displays the results as follows:

```
;; Load HTTP extension DLL
(load-extension "stc_monkhttp_nossl.dll")

;; Create an HTTP session handle
(define hCon (http-acquire-provider "jdoe" "0E0102" "" "" 0))

;; Execute the HTTP GET method
(http-get hCon "http://www.somesite.com" 0 "accept:text/*")
(define pszData (http-get-result-data hCon)

;; Print the results
(display pszData)

;; Free HTTP session handle
(http-release-provider hCon)
(set! hCon 0)
```

Parameters could be passed by this script by appending them to the URL using the **application/x-www-form-urlencoded** format, for example,

http://peterw?param1=16&param2=Lorne+Street

3.3.2 POST (Outbound) Example (HTTP_post)

The script in this section contains three examples: one posts to an ASP page, and the other two post to scripts at the specified URLs. The results are displayed.

```
;; Load HTTP extension DLL
(load-extension "stc_monkhttp_nossl.dll")

;; Create an HTTP session handle
(define hCon (http-acquire-provider "jdoe" "0E0102" "" "" 0))

;; Post to an Active Server Page (ASP) and print server reply
(define postCmd (http-post hCon "http://stingray/Project3/
Project3.asp" 0
"accept:text/*" "Content-Type: application/x-www-form-
urlencoded" "text1=doe"))
(define postData (http-get-result-data hCon))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
    (display postData)
    )
)

;; Post form data to a CGI script and print server reply
(define postCmd (http-post hCon "http://info.netscape.com/
home_search2.cgi"
0 "accept:text/*" "Content-Type: application/x-www-form-urlencoded"
"cp=Netscape&version=C&searchstring=Martin+Luther+King"))
(define postData (http-get-result-data hCon))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
    (display postData)
    )
)

;; Post form data to a CGI script and print server reply
(define postCmd (http-post hCon "http://search.netscape.com/cgi-bin/
search"
0 "accept:text/*" "Content-Type: application/x-www-form-urlencoded"
"search=Mars+missions"))
(define postData (http-get-result-data hCon))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
    (display postData)
    )
)

;; Free HTTP session handle
(http-release-provider hCon)
(set! hCon 0)
```

3.3.3 Input File based Example (AUTO_HTTP)

The sample in this section illustrates an input file for an inbound e*Way. (Change “somesite” to a valid site address.

Note: *When using an input file, it is necessary to modify the fields within the configuration file to match those within the input file, or to leave the fields blank. If a field in the configuration file, such as the* ***Request-content*** *parameter contains a string, and it does not appear within the input file, e*Gate will attempt to append the information. If within the input file, the delimiters are left empty the action within the configuration file will be used.*

The following input data is in the AUTO_HTTP schema and executes a POST or GET as specified. The following illustrates typical GET input data which might be passed to an HTTP(S) e*Way.

```
http://www.somesitea.com|GET|
http://www.somesitea.com|GET|
http://www.somesiteb.com|GET|
http://info.somesitec.com|GET|
http://finance.somesiteb.com/q|GET|s=amd&d=v1
http://finance.somesiteb.com/q|GET|s=stcs&d=v1
http://finance.somesiteb.com/q|GET|s=dell&d=v4
http://finance.somesiteb.com/q|GET|s=turf&d=b
http://www.somesited.com|GET|
http://www.somesitee.com|GET|
http://lc6.law5.hotmail.passport.com/cgi-bin/login|GET|
http://www.somesite-facts.com/
srchgrp.asp|POST|keywords=beef&stype=AND&group=ALL
http://www.msn.com|GET|
http://shop.infospace.com/cat1.htm?qvcid=539&qcat=416&nA=11|GET|
http://www.foxnews.com/video/main.sml|GET|
http://www.launch.com/music/welcome/pvn_musicvideos/?seti=1|GET|
http://www.trip.com/content/guidesandtools/0,1324,1-1,00.html|GET|
http://microsoft.com|GET|
http://www.datek.com|GET|
http://www.home.com|GET|
http://www.hotmail.com|GET|
http://www.stc.com|GET|
http://www.nutri-facts.com/
srchgrp.asp|POST|keywords=shrimp&stype=AND&group=ALL
http://www.yahoo.com|GET|
```

Chapter 4

Clear HTTP Functions

This chapter explains the Monk functions for the HTTP(S) e*Way Intelligent Adapter when using clear hyper-text transfer protocol (HTTP).

Note: *This operation does **not** provide the Secure Sockets Layer (SSL) feature. The Monk functions for the HTTP(S) e*Way with SSL are listed in* **Chapter 8**.

4.1 HTTP Functions: Introduction

The HTTP(S) e*Way's clear HTTP (without SSL) functions fall into the following categories:

- **Basic Functions**
- **HTTP Standard Functions** on page 29
- **HTTP Monk Functions** on page 35

The rest of this chapter explains the functions in these categories.

Note: *The functions explained in this chapter can only be used by the functions defined within the e*Way's configuration file for clear HTTP. None of the functions are available to Collaboration Rules scripts executed by the e*Way.*

4.2 Basic Functions

The functions in this category control the e*Way's most basic operations. For details on these functions, see **"Basic Functions" on page 104**.

4.3 HTTP Standard Functions

The current suite of HTTP Monk standard functions are:

httpnossl-ack on page 29
httpnossl-connect on page 30
httpnossl-exchange on page 30
httpnossl-init on page 31
httpnossl-nack on page 31
httpnossl-notify on page 32
httpnossl-outgoing on page 32
httpnossl-shutdown on page 33
httpnossl-startup on page 34
httpnossl-verify on page 34

httpnossl-ack

Syntax

```
(httpnossl-ack message-string)
```

Description

httpnossl-ack sends a positive acknowledgment to the external system after all Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Parameters

Name	Type	Description
message-string	String	The Event for which an acknowledgment is sent.

Return Values

String

An empty string indicates a successful operation. The e*Way is then able to proceed with the next request.

CONNERR indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.

Additional Information

See **"Positive Acknowledgment Function" on page 73** for more information.

httpnossl-connect

Syntax

```
(httpnossl-connect)
```

Description

httpnossl-connect establishes a connection to the external system.

Parameters

None.

Return Values

String
UP indicates the connection is established. Anything else indicates no connection.

Throws

None.

Additional Information

See **"External Connection Establishment Function" on page 71** for more information.

httpnossl-exchange

Syntax

```
(httpnossl-exchange)
```

Description

httpnossl-exchange sends a received event from the external system to e*Gate. The function expects no input.

Parameters

None.

Return Values

String
An empty string indicates a successful operation. Nothing is sent to e*Gate.

A message string indicates successful operation and the Event is sent to e*Gate.

CONNERR indicates a problem with the connection to the external system. When the connection is re-established this function is re-executed with the same input Event.

Throws

None.

Additional Information

See **"Exchange Data with External Function" on page 70** for more information.

httpnossl-init

Syntax

```
(httpnossl-init)
```

Description

httpnossl-init begins the initialization process for the e*Way. This function loads the **stc_monkhttp_nossl.dll** file and the initialization file, thereby making the function scripts available for future use.

Parameters

None.

Return Values

String
If a FAILURE string is returned, the e*Way shuts down. Any other return indicates success.

Throws

None.

Additional Information

Within this function, any necessary global variables to be used by the function scripts could be defined. The internal function that loads this file is called once when the e*Way first starts up.

See **"Monk Environment Initialization File" on page 68** for more information.

httpnossl-nack

Syntax

```
(httpnossl-nack message-string)
```

Description

httpnossl-nack sends a negative acknowledgment to the external system when the e*Way fails to process and queue Events from the external system.

Parameters

Name	Type	Description
message-string	String	The Event for which a negative acknowledgment is sent.

Return Values

String
An empty string indicates a successful operation.

CONNERR indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.

Throws

None.

Additional Information

See **"Negative Acknowledgment Function" on page 73** for more information.

httpnossl-notify

Syntax

```
(httpnossl-notify command)
```

Description

httpnossl-notify notifies the external system that the e*Way is shutting down.

Parameters

Name	Type	Description
command	String	When the e*Way calls this function, it passes the string SHUTDOWN_NOTIFICATION as the parameter.

Return Values

String
Returns a null string.

Throws

None.

Additional Information

See **"Shutdown Command Notification Function" on page 74** for more information.

httpnossl-outgoing

Syntax

```
(httpnossl-outgoing event-string)
```

Description

httpnossl-outgoing is used for sending a received message from e*Gate to the external system.

Parameters

Name	Type	Description
event-string	String	The Event to be processed.

Return Values

String

An empty string indicates a successful operation.

RESEND causes the Event to be immediately resent.

CONNERR indicates a problem with the connection to the external system. When the connection is re-established this function is re-executed with the same input Event.

DATAERR indicates the function had a problem processing data. If the e*Gate journal is enabled, the Event is journaled and the failed Event count is increased (the input Event is essentially skipped in this process). Use the **event-send-to-egate** function to place bad Events in a bad-event IQ. See **event-send-to-egate** on page 105 for more information on this function.

Additional Information

See **"Process Outgoing Message Function" on page 69** for more information.

httpnossl-shutdown

Syntax

```
(httpnossl-shutdown shutdown)
```

Description

httpnossl-shutdown requests that the external connection shutdown. A return value of SUCCESS indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. You then must execute a call (**shutdown-request** on page 106) from within a Monk function to allow the requested shutdown process to continue.

Parameters

Name	Type	Description
shutdown	String	When the e*Way calls this function, it passes the string SUSPEND_NOTIFICATION as the parameter.

Return Values

String

SUCCESS allows an immediate shutdown to occur. Anything else delays shutdown until the **shutdown-request** is executed successfully.

Throws

None.

Additional Information

See **"External Connection Shutdown Function" on page 72**.

httpnossl-startup

Syntax

```
(httpnossl-startup)
```

Description

httpnossl-startup is used for function loads that are specific to this e*Way and invokes startup.

Parameters

None.

Return Values

String
FAILURE causes a shutdown of the e*Way. Any other return indicates success.

Throws

None.

Additional Information

Use this function to initialize the external system before any data exchange starts. Any additional variables can be defined here.

See **"Startup Function" on page 69** for more information.

httpnossl-verify

Syntax

```
(httpnossl-verify)
```

Description

httpnossl-verify is used to verify whether the connection to the external system is established.

Parameters

None.

Return Values

String
UP or SUCCESS if the connection established. Anything other value indicates the connection is not established.

Throws

None.

Additional Information

See **"External Connection Verification Function" on page 72** for more information.

4.4 HTTP Monk Functions

The HTTP Monk functions are used to invoke contact with the HTTP Web server to upload (post) or download (get) data from it.

The Monk functions are:

http-acquire-provider on page 36

http-add-content-type-param on page 36

http-add-header on page 37

http-clear-content-type-param on page 38

http-clear-headers on page 39

http-get on page 39

http-get-error-text on page 40

http-get-last-status on page 41

http-get-result-data on page 44

http-post on page 44

http-release-provider on page 46

http-set-body-write-delay on page 46

http-set-proxy-properties on page 47

http-url-encode on page 48

Rules for *x-www-form-urlencoded* Format Encoding

For lists of reserved characters, control characters, delimiters, and symbols *not* to use when you are doing *x-www-form-urlencoded* format encoding, see the tables under **"Rules for x-www-form-urlencoded Format Encoding" on page 115**.

http-acquire-provider

Syntax

```
(http-acquire-provider username password agent proxy flags)
```

Description

http-acquire-provider performs the necessary initialization of underlying libraries and resources used during operations. This functions returns a connection-handle needed for subsequent operations.

Parameters

Name	Type	Description
username	Valid string	The name of the user performing the inquiry.
password	Encrypted-password	The valid password corresponding to the user above.
agent	Agent name	The user-agent name. This value is passed to the Web server by the client with each Web request, and it is usually used to specify the type of browser running as a client.
proxy	URL-string	A valid URL for the proxy, for example, "http://proxyname:8080" where 'proxyname' is the host, and 8080 is the port number on which the proxy server is serving requests. Specify "" (empty string) if none is used.
flags	Integer	set to 0 (reserved)

Return Values

handle
The handle associated with the HTTP session.

Throws

None.

Examples

```
(define hCon (http-acquire-provider "myusername" "0E102" "" "" 0))
```

http-add-content-type-param

Syntax

```
(http-add-content-type-param hCon content_type_name
content_type_value)
```

Description

http-add-content-type-param adds the content type parameter associated to the specified handle.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.
content_type_name	String	The name of the content type parameter to be added.
content_type_value	String	The value of the content type parameter to be added.

Return Values

Boolean

Returns **#t** (true) when successful; otherwise, returns **#f** (false) when an error occurs.

Throws

None.

http-add-header

Syntax

```
(http-add-header hCon field_name field_value)
```

Description

http-add-header adds a token value pair associated with the specified header.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.
field_name	String	The field name associated with the header being added. Some of the possible field names are: ▪ Accept ▪ Accept-Charset ▪ Accept-Encoding ▪ Accept-Language ▪ Authorization ▪ Expect ▪ From ▪ Host ▪ If-Match ▪ If-Modified-Since ▪ If-None-Match ▪ If-Range ▪ If-Unmodified-Since ▪ Max-Forwards ▪ Proxy-Authorization ▪ Range ▪ Referer
field_value	String	The field value associated with the header being added.

Return Values

Boolean

Returns **#t** (true) when successful; otherwise, returns **#f** (false) when an error occurs.

http-clear-content-type-param

Syntax

```
(http-clear-content-type-param hCon)
```

Description

http-clear-content-type-param clears the content type parameter associated with the specified handle.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.

Return Values

Boolean
Returns **#t** (true) when successful; otherwise, returns **#f** (false) when an error occurs.

Throws

None.

http-clear-headers

Syntax

```
(http-clear-headers hCon)
```

Description

http-clear-headers clears the headers associated with the specified handle.

Parameters

Name	Type	Description
hCon	opaque handle	The handle provided by **http-acquire-provider**.

Return Values

Boolean
Returns **#t** (true) when successful; otherwise, returns **#f** (false) when an error occurs.

Throws

None.

http-get

Syntax

```
(http-get hCon URL timeout accept-type)
```

Description

http-get obtains and stores the data referenced by the specified URL.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.
URL	String	The URL that the **http-get** request is to retrieve when executed.

Name	Type	Description (Continued)
timeout	Integer	A number representing the timeout in milliseconds that the client waits for a response from the server.
accept-type	String	The MIME type of the output data to be returned by the server. NOTE: Only text types are supported. Must be in the form: Accept:xxxx/xxxx. For example: "Accept:text/*."

Return Values

Boolean

Returns **#t** (true) when successful; otherwise, returns **#f** (false) when an error occurs.

Throws

None.

Additional Information

This function stores the data internally. In order to retrieve the data, the **http-get-result-data** function must be called. See **http-get-result-data** on page 44 for more information.

Examples

```
(define postCmd (http-get hCon "http://
www.somesite.com" 20000 "Accept:text/*"))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
    (display postData)
    )
)

(display pData)
```

http-get-error-text

Syntax

```
(http-get-error-text error_code)
```

Description

http-get-error-text obtains the explanation for the error code returned by **http-get-last-status**.

Parameters

Name	Type	Description
error_code	Integer	The handle error code returned by **http-get-last-status**.

Return Values

String
Returns the message associated with the error code returned by **http-get-last-status**.

Throws

None.

Additional Information

See **Table 2 on page 41** for a list of these error codes, along with a description of each.

http-get-last-status

Syntax

```
(http-get-last-status hCon)
```

Description

http-get-last-status returns the status from the last **http-get** or **http-put** call.

Parameters

Name	Type	Description
hCon	opaque handle	The handle provided by **http-acquire-provider**.

Return Values

Integer
Returns an integer corresponding to specified HTTP server status codes. See Table 2 for details.

Table 2 Server Status Return Codes

Return Value	Description	Return CodeType
-906	Cannot locate host	
-905	Connection timeout	
-904	Recover pipe line	
-903	If you want to pause a stream	
-902	Note the negative value	
-901	If we are in a select	
-900	(Not in use)	
-505	Bad protocol version	
-503	Service is not available	
-419	Proxy reauthentication required	

Table 2 Server Status Return Codes (Continued)

Return Value	Description	Return CodeType
-418	Reauthentication required	
-417	Expectation failed	
-416	Request range not satisfiable	
-415	Unsupported	
-414	Request-URI too long	
-413	Request entity too large	
-412	Precondition failed	
-411	Length required	
-409	Conflict	
-407	Proxy authentication failed	
-406	Not acceptable	
-404	Not found	
-403	Access forbidden	
-401	Unauthorized	
-1	Generic failure	
10	Response is stale	Cache
11	Revalidation failed	Cache
12	Disconnected operation	Cache
13	Heuristic expiration	Cache
14	Transformation applied	Cache
99	Cache warning	Cache
100	Continue	Information
101	Switching protocols	Information
200	OK	Success
201	Created	Success
202	Accepted	Success
203	Non-authoritative information	Success
204	Document updated	Success
205	Reset content	Success
206	Partial content	Success
207	Partial update OK	Success
300	Multiple choices	Redirection
301	Moved permanently	Redirection
302	Found	Redirection

Table 2 Server Status Return Codes (Continued)

Return Value	Description	Return CodeType
303	See other	Redirection
304	Not Modified	Redirection
305	Use proxy	Redirection
306	Proxy redirect	Redirection
307	Temporary redirect	Redirection
400	Bad request	Client_error
401	Unauthorized	Client_error
402	Payment required	Client_error
403	Forbidden	Client_error
404	Not found	Client_error
405	Method not allowed	Client_error
406	Not acceptable	Client_error
407	Proxy authentication required	Client_error
408	Request timeout	Client_error
409	Conflict	Client_error
410	Gone	Client_error
411	Length required	Client_error
412	Precondition failed	Client_error
413	Request entity too large	Client_error
414	Request-URI too large	Client_error
415	Unsupported media type	Client_error
416	Range not satisfiable	Client_error
417	Expectation failed	Client_error
418	Reauthentication required	Client_error
419	Proxy reauthentication required	Client_error
500	Internal server error	Server_error
501	Not implemented	Server_error
502	Bad gateway	Server_error
503	Service unavailable	Server_error
504	Gateway timeout	Server_error
505	HTTP version not supported	Server_error
506	Partial update not implemented	Server_error

Note: *See the HTTP server documentation for more information.*

http-get-result-data

Syntax

```
(http-get-result-data hCon)
```

Description

http-get-result-data retrieves the data returned by the server from the last **http-get** call.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.

Return Values

String
The string contains the data requested.

Additional Information

Verify the success of the **http-get** or **http-post** function, prior to calling **http-get-result-data**.

Note: *The function must be passed as a handle that is returned from **http-acquire-provider**. The return value is valid **only** when called after a FORM get as shown in the example in this section (via **http-get**).*

Examples

```
(define postCmd (http-post hCon "http://stingray/Project3/
Project3.asp" 0
"accept:text/*" "Content-Type: application/x-www-form-
urlencoded" "test1=hello&test2=world"))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
    (display postData)
    )
)
```

http-post

Syntax

```
(http-post hCon URL timeout accept-string content-type post-data)
```

Description

http-post posts to a specified URL. The post request submits data to a form.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.
URL	String	The URL to which the data is posted.
timeout	Integer	A number representing the timeout in milliseconds that the client waits for a response from the server.
accept-string	String	The MIME type of the output data to be returned by the server. NOTE: Only text types are supported. Must be in the form: accept:xxxx/xxxx For example: “Accept:text/*.”
content-type	String	Content type of the data passed to the post-data parameter. The default: application/x-www-form-urlencoded.
post-data	String	Defines the encoded value to pass to the Web server as part of a POST request. The example here is encoded in the default "application/x-www-form-urlencoded" scheme. (stringx=data_string&stingy=data_string) example:”`test1=hello&test2=world`”

Return Values

Boolean

If successful, returns **#t** (true); otherwise, returns **#f** (false).

Throws

None.

Additional Information

Verify the successful result of the **http-post** call before calling **http-get-result-data**.

For more information on acceptable format types, see **“Rules for x-www-form-urlencoded Format Encoding” on page 115**.

When the Web server sends a “cookie” to the e*Way, the e*Way stores it away in memory. Each time the e*Way needs to “Post” to the same Web site, it references the same cookie as received initially (usually the login page). The e*Way is able to store cookie “A” for one site, cookie “B” for another site, etc., and associates each cookie with the relevant site.

Example One

```
(define postCmd (http-post hCon "http://stingray/Project3/
Project3.asp" 0
"accept:text/*" "Content-Type: application/x-www-form-
urlencoded" "test1=hello&test2=world"))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
```

```
        (display postData)
        )
    )
```

Example Two

```
1st eWay post ---> login page
    <---login page responds with cookie "A"
2nd eWay post (with cookie "A" ---> next page
```

http-release-provider

Syntax

```
(http-release-provider hCon)
```

Description

http-release-provider de-allocates the HTTP session handle obtained from **http-acquire-provider**.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.

Return Values

None.

Throws

None.

http-set-body-write-delay

Syntax

```
(http-set-body-write-delay hCon <1stdelay> <2nddelay>)
```

Description

http-set-body-write-delay is used to specify the delays added before writing the HTTP body data. See the following Web site for a detailed explanation:

http://www.w3.org/Library/src/HTTP.html

Parameters

Name	Type	Description
hCon	Handle	The handle from **http-acquire-provider** (see **http-acquire-provider** on page 36).

Name	Type	Description
1stdelay	Integer	The desired first delay time in milliseconds (default is 2000 ms).
2nddelay	Integer	The desired second delay time in milliseconds (default is 3000 ms).

Return Values

None.

Throws

None.

http-set-proxy-properties

Syntax

```
(http-set-proxy-properties hCon proxyUrl port proxyUser
proxyPassword)
```

Description

http-set-proxy-properties defines the parameters necessary to access the proxy server.

Parameters

Name	Type	Description
hCon	Opaque handle	The handle provided by **http-acquire-provider**.
proxyUrl	String	The proxy URL, for example: “www.somesite.com” or “www.somesite.com:8080”
port	Integer	The port number on which the proxy server is listening.
proxyUser	String	A valid user name.
proxyPassword	String	An encrypted password associated with the above named user. Use the **encrypt-password** function to create this password. See the *Monk Developer’s Reference* for more information.

Return Values

Boolean

If successful, returns **#t** (true); otherwise, returns **#f** (false).

Throws

None.

http-url-encode

Syntax

```
(http-url-encode input_data)
```

Description

http-url-encode encodes the given string into x-*www-form-urlencoded* format.

Parameters

Name	Type	Description
input_data	String	The string to be encoded.

Return Values

String
Returns the encoded string.

Throws

None.

Additional Information

In previous releases of the HTTP(S) e*Way this was handled automatically. Currently, this function must be called in order to transform the data string into a URL-encoded format.

Chapter 5

Secure Sockets Layer Operation

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

5.1 Using Secure Sockets Layer: Overview

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

You can use the **openssl** utility program to generate certificate and key files to use with the HTTP(S) e*Way. If your client certificates are in the PKCS12 format, you must convert them to PEM-formatted files. The HTTP(S) e*Way only uses PEM files.

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and open-source toolkit implementing SSL versions 2.0 and 3.0, and Transport Layer Security (TLS) version 1.0 protocols, as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

For more information, see the following Web site:

http://www.openssl.org/

5.2 Certificates and Security

The HTTP(S) e*Way uses certificates to ensure the security of each transaction. Certificates are files that contain information that identifies the user or organization that owns the certificate, the period of time for which the certificate is valid, the organization that issued the certificate, and a digital “signature” that verifies the organization’s identity. Certificates are issued by a certification authority (CA), a third party that each participant in the data-exchange process trusts to verify identity and to issue appropriate certificates.

An easy way to understand certificates is to compare them to passports. Border-control authorities and citizens both agree that the agency that issues passports (the government) has the authority to do so. Each passport identifies its owner; each passport has an expiration date. Anti-counterfeiting measures built into the passport identify genuine, authorized documents.

Using certificates, the client system (in an e*Gate implementation, the HTTP(S) e*Way) is able to verify the identity of the Web server; likewise, the Web server is able to verify the identity of the client. Once both systems have verified each other's identity, a secure channel is established, and confidential information can be exchanged safely.

Important: *There must be a valid certificate located in the specified directory before a CA certificate can be authenticated. If there are no certificates located in the specified directory, if the directory load fails, or if the file within the directory is empty, the authentication process cannot proceed.*

5.3 Using the openssl Utility

This section explains how to use the **openssl** Utility with the HTTP(S) e*Way, to convert PKCS12-formatted files.

5.3.1 Working with PKCS12 files

Run the following command at the command line:

```
openssl pkcs12 <options>
```

For the usage banner:

```
openssl pkcs12 -v
```

To get the CA certificate

```
openssl pkcs12 -info -in existing_cert_file.pfx -out
   new_cert_file.cer -cacerts
```

Example

```
openssl pkcs12 -info -in D:\myown.pfx -out myownCA.cer -cacerts
```

5.3.2 Converting PKCS12 Files to PEM Files

To convert a PKCS12 certificate and key file to two separate PEM files

Get the client certificate PEM file as follows:

```
openssl pkcs12 -info -in D:\myown.pfx -out myownClient.cer -clcerts
   -nokeys
```

Get the client key PEM file as follows:

```
openssl pkcs12 -info -in D:\myown.pfx -out myownKey.cer -nocerts
   -nodes
```

5.3.3 Converting DER Files to PEM Files

To convert a DER certificate and key file to two separate PEM files

```
openssl x509 -inform DER -in SoCoCert.der.cer -outform PEM -out
    SoCoCert.pem
```

Import a CA certificate to the truststore as follows:

```
keytool -import -keystore trustcacertsjks -file SoCoCert.der.cer
    alias pantallos
```

or

```
keytool -import -keystore trustcacertsjks -file SoCoCert.pem -alias
    pantallos
```

The keytool utility understands both formats. The password for **trustcacertsjks** is **seebeyond** (all in lowercase). Set this file as the truststore and **seebeyond** as the truststore password for the e*Way configuration.

5.3.4 Converting Other Formats

For more information on using **openssl** for converting other certificate formats to PEM, see the following Web site:

http://www.openssl.org/docs/apps/openssl.html

5.4 SSL Handshaking

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

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

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

- **Figure 4 on page 52** shows a diagram of the SSL handshake dialog for server-side authentication.
- **Figure 5 on page 53** shows a diagram of the SSL handshake dialog for dual authentication.

Figure 4 Server-side Authentication

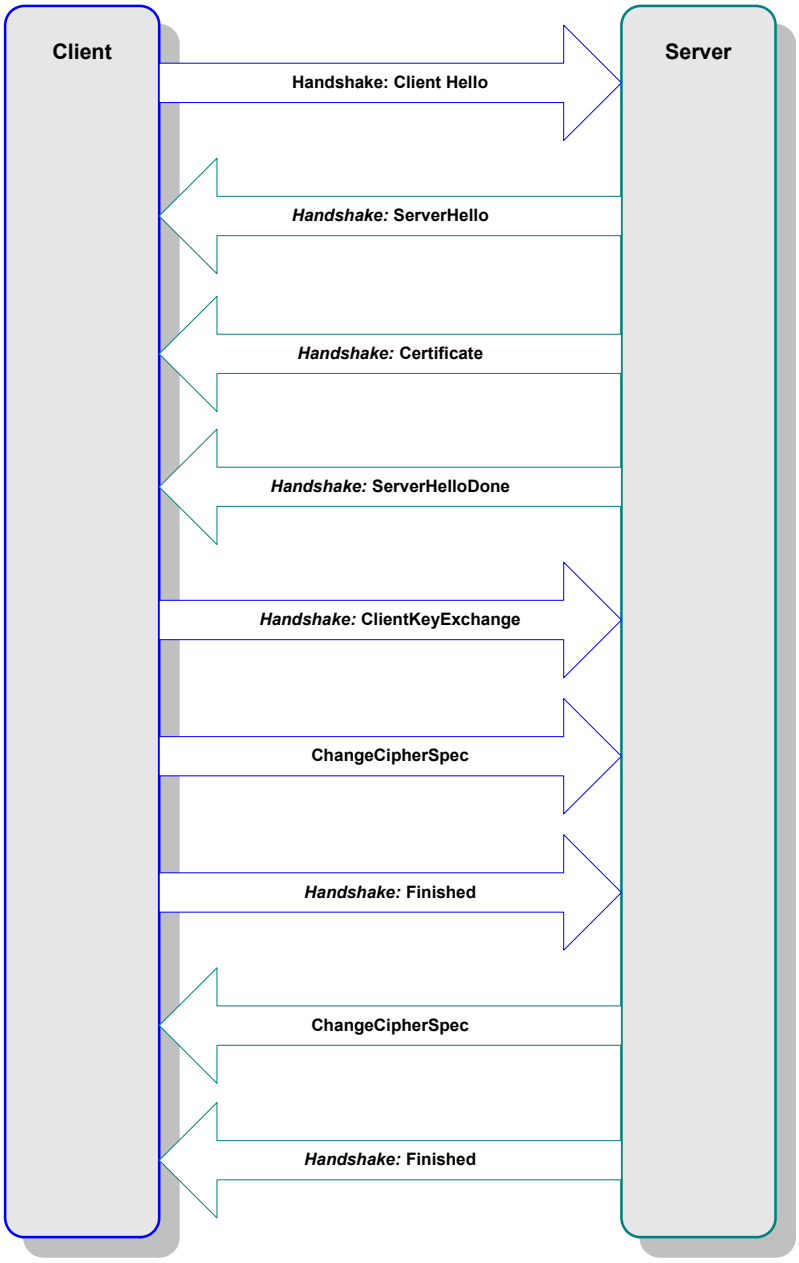


Figure 5 Dual Authentication

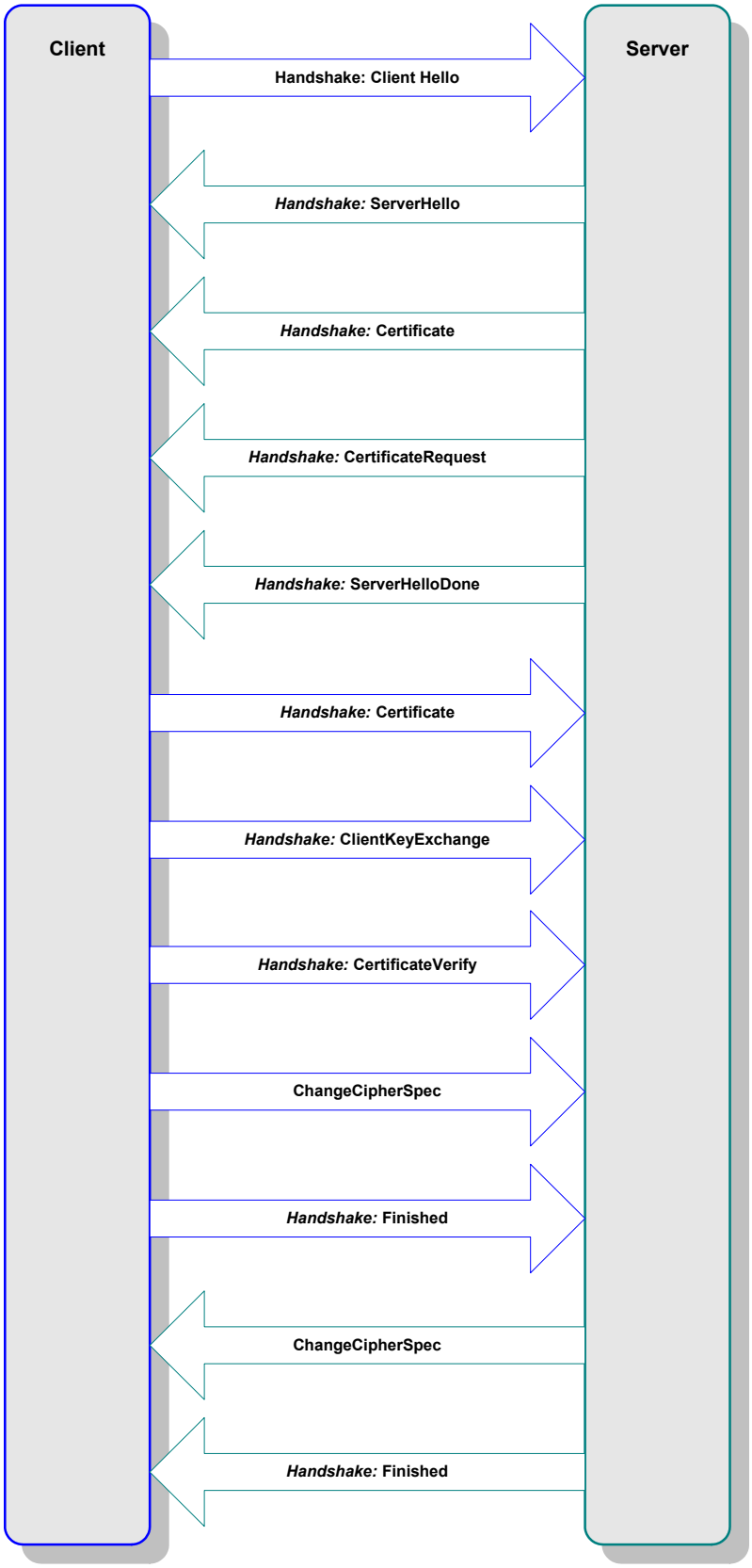
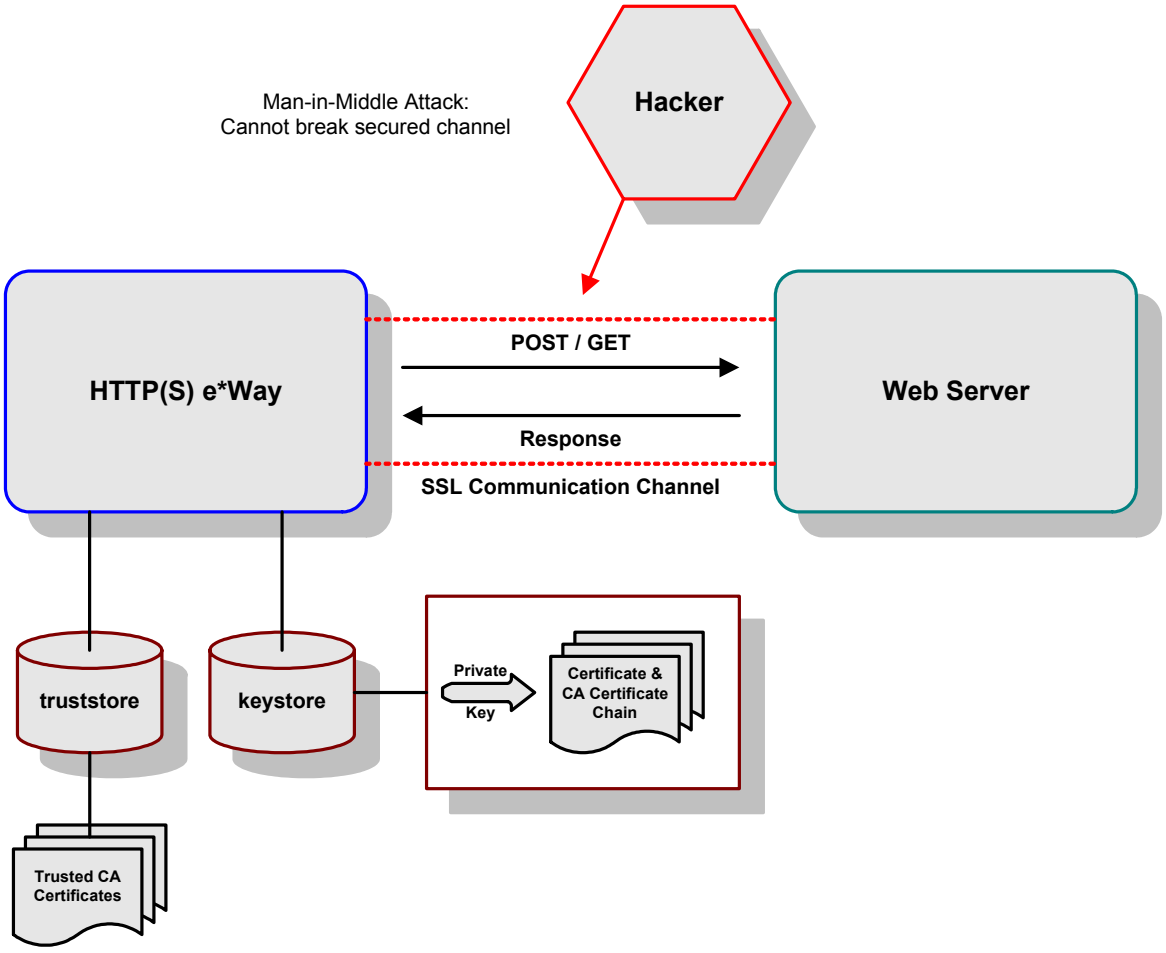


Figure 6 shows a diagram of general SSL operation with the HTTP(S) e*Way.

Figure 6 General SSL Operation: HTTP(S) e*Way



Chapter 6

HTTP(S) e*Way Configuration

This chapter describes how to configure the HTTP(S) e*Way Intelligent Adapter.

6.1 Introduction

This chapter describes the procedure for configuring a new HTTP(S) e*Way. You can also modify this procedure to use existing e*Ways. e*Way configuration parameters are set using the e*Way editor. Procedures for creating and editing e*Gate components are provided in the Schema Designer's online help.

Before you can run the HTTP(S) e*Way, you must configure it using the e*Way Editor, which is accessed from the e*Gate Schema Designer GUI. The HTTP(S) e*Way package includes a default configuration file which you can modify using this editor.

6.2 e*Way Configuration Parameters

Start with the e*Gate Schema Designer graphical user interface (GUI) to set or change an e*Way's configuration parameters.

To set or change e*Way configuration parameters

1. In the Schema Designer's Component Editor pane, 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 the following sections:

- **"General Settings" on page 56**
- **"Communication Setup" on page 57**
- **"Monk Configuration" on page 60**
- **"HTTP Configuration" on page 74**
- **"HTTP Proxy Configuration" on page 77**
- **"HTTP(S) Configuration" on page 78**

6.2.1 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid file name, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is stored in the e*Gate **SystemData** directory. See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

Additional Information

An Event is journaled for the following conditions:

- When the number of resends is exceeded (see **Max Resends Per Message** in the next section)
- When its receipt is due to an external error, but **Forward External Errors** is set to **No**. (See **"Forward External Errors" on page 57** for more information.)

Max Resends Per Message

Description

Specifies the maximum number of times the e*Way attempts to resend a message to the external system after receiving an error.

Required Values

An integer between 1 and 1024. The default is 5.

Max Failed Messages

Description

Specifies the maximum number of failed messages that the e*Way allows. When the specified number of failed messages is reached, the e*Way shuts down and exits.

Required Values

An integer between 1 and 1024. The default is 3.

Forward External Errors

Description

Selects whether error messages that begin with the string **DATAERR** that are received from the external system are queued to the e*Way's configured Intelligent Queue (IQ). See **"Exchange Data with External Function" on page 70** for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages are not forwarded.

See **"Schedule-driven Data Exchange Functions" on page 65** for information about how the e*Way uses this function.

6.2.2 Communication Setup

The Communication Setup parameters control the schedule by which the e*Way obtains data from the external system.

Note: *The schedule you set using the e*Way's properties in the Schema Designer controls when the e*Way executable runs. The schedule you set within the parameters discussed in this section (using the e*Way Editor) determines when data is exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Exchange Data Interval

Description

Specifies the number of seconds the e*Way waits between calls to the **Exchange Data with External** function during scheduled data exchanges.

Required Values

An integer between 0 and 86,400. The default is 120.

Additional Information

If **Zero Wait Between Successful Exchanges** is set to **Yes** and the **Exchange Data with External Function** returns data, The **Exchange Data Interval** setting is ignored, and the e*Way invokes the **Exchange Data with External Function** immediately.

If this parameter is set to zero, there is no exchange data schedule set and the **Exchange Data with External Function** is never called.

See **"Down Timeout" on page 59** and **"Stop Exchange Data Schedule" on page 59** for more information about the data exchange schedule.

Zero Wait Between Successful Exchanges

Description

Selects whether to initiate data exchange processes after the **Exchange Data Interval** or immediately after a successful previous exchange.

Required Values

Yes or **No**. If this parameter is set to **Yes**, the e*Way immediately invokes the **Exchange Data with External** function, if the previous exchange function returned data. If this parameter is set to **No**, the e*Way always waits the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External** function. The default is **No**.

See **"Exchange Data with External Function" on page 70** for more information.

Start Exchange Data Schedule

Description

Establishes the schedule to invoke the e*Way's **Exchange Data with External** function.

Required Values

One of the following:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds)

Also required: If you set a schedule using this parameter, you must also define all of the following parameters:

- **Exchange Data with External Function** on page 70
- **Positive Acknowledgment Function** on page 73
- **Negative Acknowledgment Function** on page 73

If you do not do so, the e*Way terminates its execution when the schedule attempts to start.

Since months do not all contain equal numbers of days, be sure not to provide boundaries that would cause an invalid date selection (for example, the 30th of every month does not include February).

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an acknowledgement (ACK) or non-acknowledgement (NAK) to the external system (using the **Positive** and **Negative Acknowledgment** functions) and whether the connection to the external system is active.

If no ACK/NAK is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External** function. Thereafter, the **Exchange Data with External** function is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

See **"Exchange Data with External Function" on page 70**, **"Exchange Data Interval" on page 57**, and **"Stop Exchange Data Schedule" on page 59** for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

  Since months do not all contain equal numbers of days, be sure not to provide boundaries that would cause an invalid date selection (for example, the 30th of every month does not include February).

Down Timeout

Description

Specifies the number of seconds that the e*Way waits between calls to the **External Connection Establishment Function**. See **"External Connection Establishment Function" on page 71** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Up Timeout

Description

Specifies the number of seconds the e*Way waits between calls to the **External Connection Verification Function**. See **"External Connection Verification Function" on page 72** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Resend Timeout

Description

Specifies the number of seconds the e*Way waits between attempts to resend a message to the external system, after receiving an error message.

Required Values

An integer between 1 and 86,400. The default is 10.

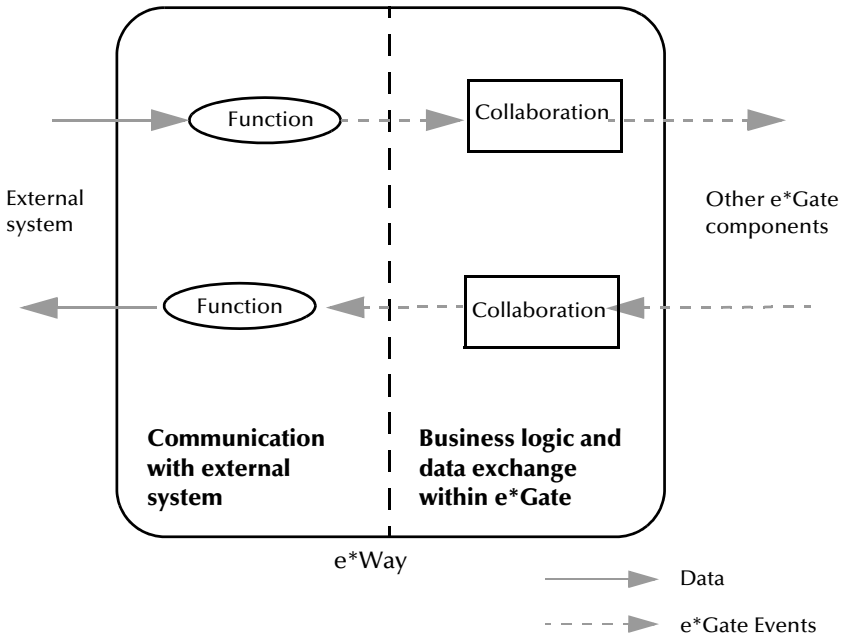
6.2.3 Monk Configuration

The parameters in this section help you set up the information required by the e*Way to utilize Monk for communication with the external system.

e*Way Structure

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 7) handles communication with the external system. The other half manages the Collaborations that process data and subscribe or publish to other e*Gate components.

Figure 7 e*Way Internal Architecture



The communications half of the e*Way uses Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate Events and send those Events to Collaborations, and manage the connection between the e*Way and the external system.

The **Monk Configuration** options explained in this section control the Monk environment and define the Monk functions used to perform these basic e*Way operations. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as, Notepad, or UNIX vi).

The communications half of the e*Way is single-threaded. Functions run serially, and only one function can be executed at a time. The business logic side of the e*Way is multi-threaded, with one executable thread for each Collaboration. Each thread maintains its own Monk environment. Therefore, information such as variables, functions, path information, and so on, cannot be shared between threads.

Operational Details

Table 3 shows the categories of Monk functions in the communications half of an e*Way.

Table 3 Categories of Communication Monk Functions

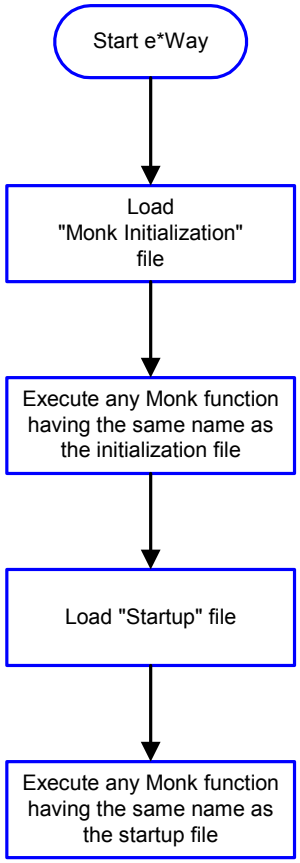
Type of Operation	Name
Initialization	**Startup Function** on page 69 (also see **Monk Environment Initialization File** on page 68)
Connection	**External Connection Establishment Function** on page 71 **External Connection Verification Function** on page 72 **External Connection Shutdown Function** on page 72
Schedule-driven data exchange	**Exchange Data with External Function** on page 70 **Positive Acknowledgment Function** on page 73 **Negative Acknowledgment Function** on page 73
Shutdown	**Shutdown Command Notification Function** on page 74
Event-driven data exchange	**Process Outgoing Message Function** on page 69

A series of figures on the next several pages illustrates the interaction and operation of these functions.

Initialization Functions

Figure 8 on page 62 illustrates how the e*Way executes its initialization functions.

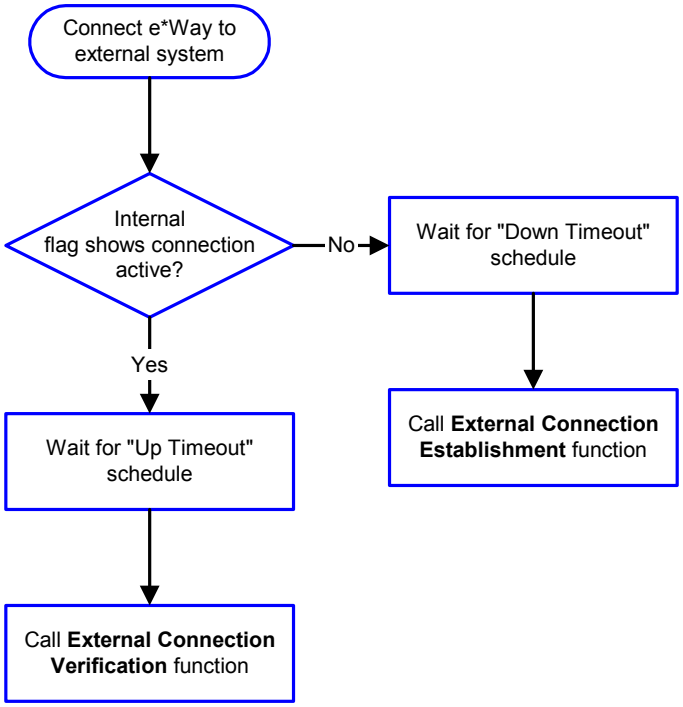
Figure 8 Initialization Functions



Connection Functions

Figure 9 on page 63 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 9 Connection Establishment and Verification Functions

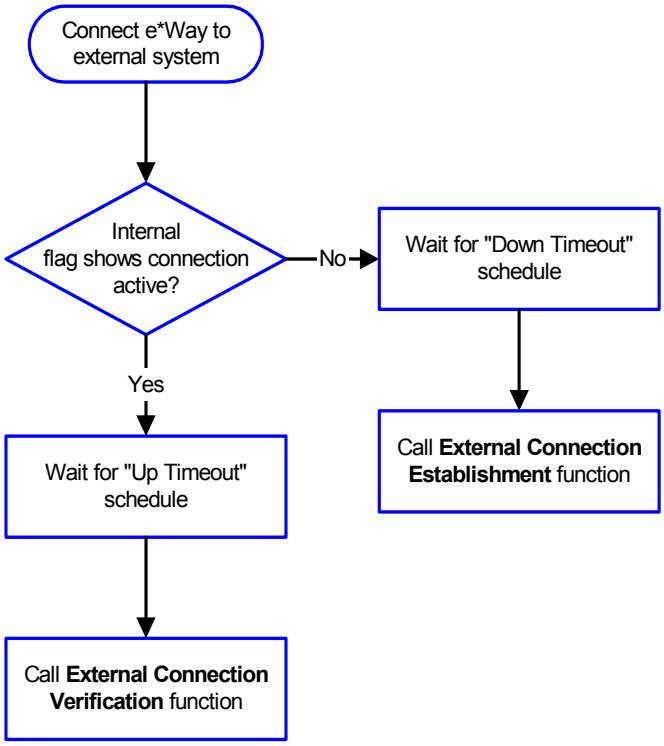


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 11 on page 65** *and* **Figure 13 on page 67** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **HTTP Standard Functions** *on page 108 and* **send-external-down** *on page 106 for more information.*

Figure 10 illustrates how the e*Way executes its "connection shutdown" function.

Figure 10 Connection Shutdown Function



Schedule-driven Data Exchange Functions

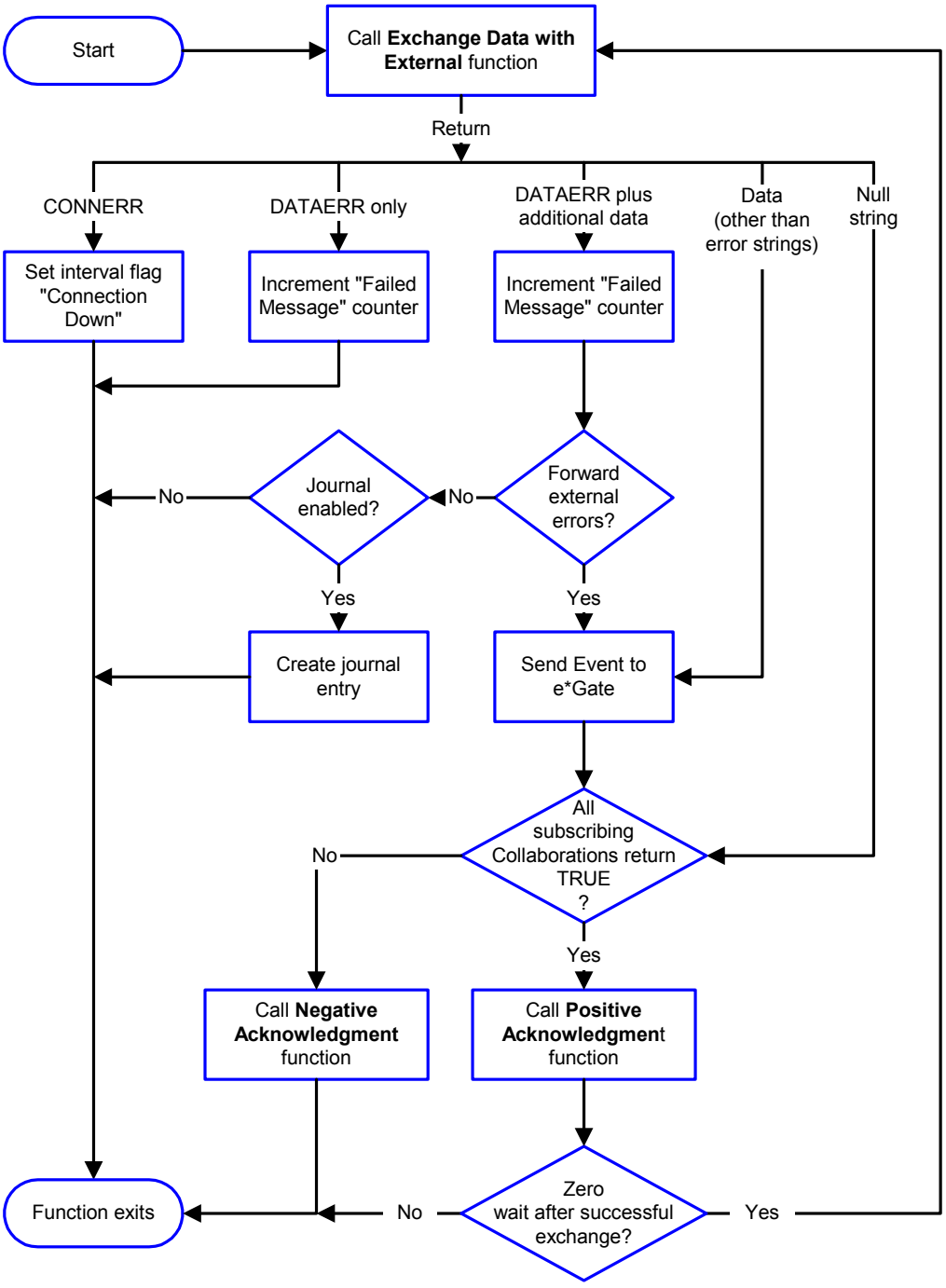
Figure 11 on page 65 illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgment Function** and **Negative Acknowledgment Function** are also called during this process.

Start can occur in any of the following ways:

- The **Start Data Exchange** time occurs
- Periodically during data-exchange schedule (after **Start Data Exchange** time, but before **Stop Data Exchange** time), as set by the **Exchange Data Interval**
- The **start-schedule** Monk function is called

After the function exits, the e*Way waits for the next **start-schedule** time or command.

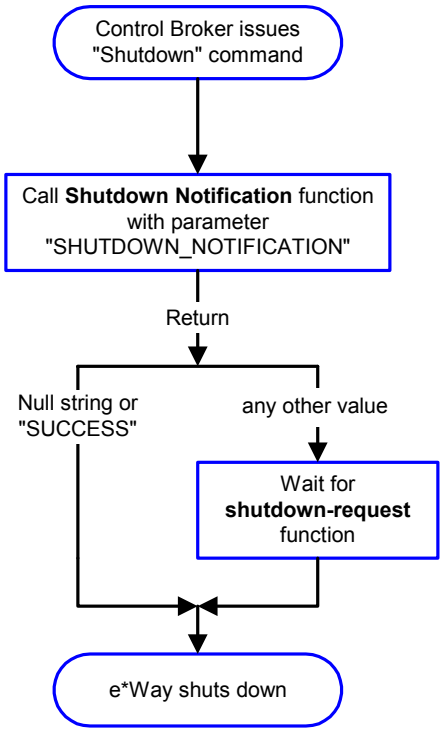
Figure 11 Schedule-driven Data Exchange Functions



Shutdown Functions

Figure 12 illustrates how the e*Way implements the **shutdown request** function.

Figure 12 Shutdown Functions



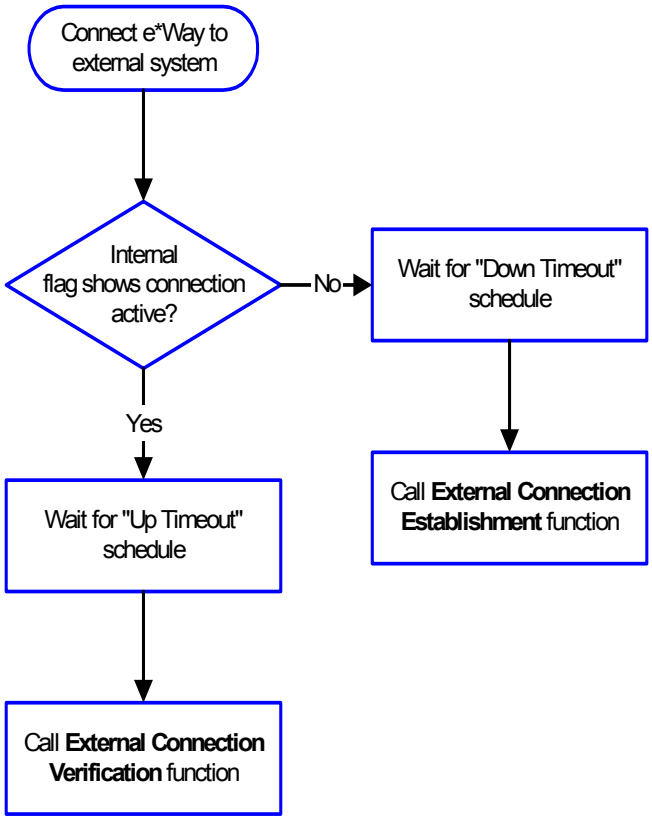
Event-driven Data Exchange Functions

Figure 13 on page 67 illustrates event-driven data exchange using the **Process Outgoing Message Function**.

Every two minutes, the e*Way checks the **Failed Message** counter against the value specified by the **Max Failed Messages** parameter. When the **Failed Message** counter exceeds the specified maximum value, the e*Way logs an error and shuts down.

After the function exits, the e*Way waits for the next outgoing Event.

Figure 13 Event-driven Data-exchange Functions



How to Specify Function Names or File Names

Parameters that require the name of a Monk function accept either a function name or a file name. If you specify a file name, be sure that the file has one of the following extensions:

- **.monk**
- **.tsc**
- **.dsc**

Additional Path

Description

Specifies a path to be added to the *load path*, the path Monk uses to locate files and data (set internally within Monk). The directory specified in **Additional Path** is searched before the default load path.

Required Values

A path name, or a series of paths separated by semicolons. This parameter is optional and may be left blank.

Additional information

The default load paths are determined by the **bin** and **Shared Data** settings in the **.egate.store** file. See the *e*Gate Integrator System Administration and Operations Guide* for more information about this file.

To specify multiple directories, manually enter the directory names rather than selecting them with the **File Selection** button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths, for example:

```
monk_scripts\my_dir;c:\my_directory
```

The internal e*Way function that loads this path information is called only once, when the e*Way first starts up.

Auxiliary Library Directories

Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories are automatically loaded into the e*Way's Monk environment.

Required Values

A path name, or a series of paths separated by semicolons. The default is **monk_library/ewhttp**.

Additional information

To specify multiple directories, manually enter the directory names rather than selecting them with the **File Selection** button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths, for example:

```
monk_scripts\my_dir;c:\my_directory
```

This function is called once when the e*Way first starts up. This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which are loaded after the auxiliary library directories are loaded. Typically, it is a good practice to initialize any global Monk variables that can be used by any other Monk extension scripts.

Required Values

A file name within the load path, or file name plus path information (relative or absolute). If path information is specified, that path is appended to the load path. See **"Additional Path" on page 67** for more information about the load path. The default is **http-init.monk** (see **http-init** on page 110 for more information).

Additional information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way loads this file and tries to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way attempts to execute the function **my-init**).

Typically, it is a good practice to initialize any global Monk variables that may be used by any other Monk extension scripts.

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 8 on page 62**).

Startup Function

Description

Specifies a Monk function that the e*Way loads and invokes upon startup or whenever the e*Way's configuration changes before it enters into its initial communication state. This function is used so that the external system can be initialized before the message exchange starts.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. (The default is **http-startup**. See **http-startup** on page 113 for more information.)

Additional information

The function accepts no input, and must return a string.

The string FAILURE indicates that the function failed; any other string (including a null string) indicates success.

This function is called after the e*Way loads the specified **Monk Environment Initialization file** and any files within the specified **Auxiliary Directories**.

The e*Way loads this file and tries to invoke a function of the same base name as the file name (see **Figure 8 on page 62**). For example, for a file named **my-startup.monk**, the e*Way attempts to execute the function **my-startup**.

Process Outgoing Message Function

Description

Specifies the Monk function responsible for sending outgoing messages (Events) from the e*Way to the external system. This function is event-driven (unlike the Exchange Data with External function, which is schedule-driven).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***You may not leave this field blank.*** (The default is **http-outgoing**. See **http-outgoing** on page 111 for more information.)

Additional Information

The function requires a non-null string as input (the outgoing Event to be sent) and must return a string.

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Schema Designer). The function returns one of the following (see **Figure 13 on page 67** for more details):

- **Null string:** Indicates that the Event was published successfully to the external system.
- **RESEND:** Indicates that the Event should be resent.
- **CONNERR:** Indicates that there is a problem communicating with the external system.
- **DATAERR:** Indicates that there is a problem with the message (Event) data itself.
- **Any other string:** If a string other than one of the previous is returned, the e*Way creates an entry in its log file indicating that an attempt has been made to access an unsupported function.

Note: *If you wish to use **event-send-to-egate** to enqueue failed Events in a separate IQ, the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See* **event-send-to-egate** *on page 105 for more information.*

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven).

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. (The default is **http-exchange**. See **http-exchange** on page 109.)

Additional Information

The function accepts no input and must return a string (see **Figure 11 on page 65** for more details):

- **Null string:** Indicates that the data exchange was completed successfully. No information is sent into the e*Gate system.
- **CONNERR:** Indicates that a problem with the connection to the external system has occurred.
- **DATAERR:** Indicates that a problem with the data itself has occurred. The e*Way handles the string "DATAERR" and "DATAERR" plus additional data differently; see **Figure 11 on page 65** for more details.
- **Any other string:** The contents of the string are packaged as an inbound Event. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

This function is initially triggered by the **Start Data Exchange** schedule or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been acknowledged or not acknowledged (by the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter.

If this parameter is set to **Yes**, the e*Way immediately calls the **Exchange Data with External** function again. Otherwise, the e*Way does not call the function until the next scheduled **start-exchange** time, or the schedule is manually invoked using the Monk function **start-schedule** (see **start-schedule** on page 107 for more information).

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way calls when it has determined that the connection to the external system is down.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. *This field cannot be left blank*. (The default is **http-connect**. See **http-connect** on page 109 for more information.)

Additional Information

The function accepts no input and must return a string.

- "SUCCESS" or "UP": Indicates that the connection was established successfully.
- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Down Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Verification** function (see below) is called when the e*Way has determined that its connection to the external system is up.

External Connection Verification Function

Description

Specifies a Monk function that the e*Way calls when its internal variables show that the connection to the external system is up.

Required Values

The name of a Monk function. This function is optional. If no **External Connection Verification** function is specified, the e*Way executes the **External Connection Establishment** function in its place. The default is **http-verify** (see **http-verify** on page 113 for more information).

Additional Information

The function accepts no input and must return a string as follows:

- **SUCCESS or UP:** Indicates that the connection was established successfully.
- **Any other string:** Including the null string, indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Up Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Establishment** function (see the previous paragraphs) is called when the e*Way has determined that its connection to the external system is down.

External Connection Shutdown Function

Description

Specifies a Monk function that the e*Way calls to shut down the connection to the external system.

Required Values

The name of a Monk function. (The default is **http-shutdown**. See **http-shutdown** on page 112 for more information.)

Additional Information

This function requires a string as input, and may return a string.

This function is only invoked when the e*Way receives a "suspend" command from a Control Broker. When the **suspend** command is received, the e*Way invokes this function, passing the string SUSPEND_NOTIFICATION as an argument.

Any return value indicates that the **suspend** command can proceed and that the connection to the external system can be broken immediately.

Note: *Include in this function any required clean-up operations that must be performed as part of the shutdown procedure, but before the e*Way exits.*

Positive Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls when *all* the Collaborations to which the e*Way has sent data have processed and enqueued that data successfully.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined. (The default is **http-ack**. See **http-ack** on page 108 for more information.)

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string as follows:

- **CONNERR:** Indicates a problem with the connection to the external system. When the connection is re-established, the **Positive Acknowledgment** function is called again, with the same input data.
- **Null string:** The function completed its execution successfully.

After the **Exchange Data with External** function returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the **Positive Acknowledgment** function. Otherwise, the e*Way executes the **Negative Acknowledgment** function.

Note: *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

Negative Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls when the e*Way fails to process and enqueue Events from the external system.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined. See **http-nack** on page 110 for more information)

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string as follows:

- **CONNERR:** Indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.
- **Null string:** The function completed its execution successfully.

This function is only called during the processing of inbound Events. After the **Exchange Data with External** function returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is not completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the Negative Acknowledgment function (otherwise, the e*Way executes the Positive Acknowledgment function).

Note: *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

Shutdown Command Notification Function

Description

Specifies a Monk function that is called when the e*Way receives a **shutdown** command from the Control Broker. This parameter is optional.

Required Values

The name of a Monk function.

Additional Information

When the Control Broker issues a **shutdown** command to the e*Way, it calls this function with the string SHUTDOWN_NOTIFICATION passed as a parameter.

The function accepts a string as an input and must return a string as follows:

- **A null string or SUCCESS:** Indicates that the shutdown can occur immediately.
- **Any other string:** Indicates that shutdown must be postponed. Once postponed, shutdown does not proceed until the Monk function **shutdown-request** is executed (see **shutdown-request** on page 106).

Note: *If you postpone a shutdown using this function, be sure to use the* ***shutdown-request*** *function to complete the process in a timely manner.*

6.2.4 HTTP Configuration

This section defines the hyper-text transfer protocol (HTTP) parameters used in the **http-acquire-provider** (see **http-acquire-provider** on page 117 for more information), as well as the **GET** and **POST** calls (see **Sample Configurations** on page 87 for more information).

Request

Description

Specifies whether this request is to use the **GET** or **POST** method.

Required Values

GET or **POST**.

Timeout

Description

Specifies the amount of time in milliseconds the e*Way waits for a response from the Web server.

Required Values

An integer between **1** and **864000**. The default is **50000**.

URL

Description

Specifies the target URL for the **GET** or **POST** command. Your target URL must process the **POST** data or **GET** request.

Required Values

A string containing a valid URL. The URL must be complete, as in the examples below:

```
HTTP(S)://www.yourcompany.com:2080
```

or

```
HTTP(S)://www.yourcompnay.com/search2.cgi
```

Additional Information

If you are using **GET**, you can provide parameters using the *application/x-www-form-urlencoded* notation, for example:

```
http://www.peterw.com/search?p1+fort&p2=william&p3=levack
```

Whether you need to express **GET** method parameters using the application x-*www-form-urlencoded* notation is dependent on whether the interfacing Web program requires the data to be encoded in this manner before receiving it.

User Name

Description

Specifies the user name for authentication purposes necessary for connecting to the Web server.

Required Values

A string containing any valid user name. (See also **"Encrypted Password" on page 76**)

Additional Information

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

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

Encrypted Password

Description

Specifies the encrypted password connected to the username entered previously, necessary to complete authentication.

Required Values

A string containing the valid encrypted password associated with the user name.

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

Agent

Description

Specifies an agent name to pass to the Web server. This is an arbitrary name identifying the e*Way to the Web server.

Required Values

A string. (The configured default is **e*Gate HTTP(S) e*Way**.)

Content-type

Description

Specifies the content-type of the application data.

Required Values

A string.

Additional Information

Normally, the format below is sufficient to support most applications:

Content-Type: application/x-www-form-urlencoded.

Important: *Do not change this parameter without a specific need to do so. In previous releases of the HTTP(S) e*Way this was performed automatically. With this release it is necessary to call* **http-url-encode** *on page 126.*

Request-content

Description

Specifies the content to be used with the **POST** method.

Required Values

A string. The expected string must follow the "**stringx=string_data**" format. See below for an example.

Additional Information

This parameter is ignored when the **GET** method is used.

The content is normally in the following format *application/x-www-form-urlencoded* of name/value pairs, for example:

```
p1=peterw&p2=walklett
```

Accept-type

Description

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

Required Values

A string, for example, **accept:text/***.

6.2.5 HTTP Proxy Configuration

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

Use Proxy Server

Description

Specifies whether the e*Way uses the parameter values in this section to connect through a proxy server. Select **YES** if the e*Way must connect through a proxy server, or **NO** to use a direct connection.

Required Values

YES or **NO**.

User Name

Description

Specifies the user name necessary for authentication to access the proxy server.

Required Values

A valid user name.

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

Encrypted Password

Description

Specifies the encrypted password corresponding to the user name specified previously.

Required Values

The appropriate password.

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

Server Address

Description

Specifies the URL address of the proxy server.

Required Values

A valid URL. For example:

http://myproxy

Important: *Do not specify a port number as part of the URL. Specify port number within the* ***Port Number*** *parameter.*

Port Number

Description

Specifies the port number to which the proxy server is listening.

Required Values

An Integer between 1 and 864000. The default is **8080**.

6.2.6 HTTP(S) Configuration

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

Trusted CA Certificates Directory

Description

Specifies the directory located within the e*Gate Registry in which all of the certification authority (CA) certificates are located. These certificates are used to verify a trust relationship between the user and the CA.

Required Values

A relative path name. The default is **pkicerts/trustedcas**.

Note: *All certificates in the certificate directory must be valid. Do not allow any other files to reside in this directory.*

Use Client Certificate Map

Description

Specifies whether the e*Way selects client certificates based on certificate mapping. A *certificate map* is a text file that maps a base URL to a client certificate file and client private-key file. **No** disables this feature.

Required Values

Yes or **No**.

Client Certificate Map File

Description

Specifies the directory and file name of the text file containing the client certificate map.

Required Values

A string. The default is **pkicerts/client/certmap.txt**. The string contains four fields, separated by the pipe symbol ("|"), containing the following information:

- Base URL
- Logical path and file name of the client certificate
- Logical path and file name of the client key
- Encoding type for cert and key (PEM)

Example

```
www.stc.com|pkicerts/client/certs/mycert1.cer|pkicerts/client/keys/
mycert1.key|PEM
*|pkicerts/client/certs/myglobal.cer|pkicerts/client/keys/
myglobal.key|PEM
```

Additional Information

If there is an '*' in the first column replacing the base URL, it means "all others."

If there is a '#' in the first column, the line is treated as a comment.

Lines in the file are processed from top to bottom, and the first base-URL match found is the one used.

6.3 Working with Certificates

Before the HTTP(S) e*Way can establish secure communications with an external system, the appropriate certificates must be obtained and committed to the e*Gate Registry. *Certificates* are files that contain identification information that the e*Way requires to establish a secure and trusted connection (see the **"Introduction" on page 7**" of this manual for more information about certificates).

6.3.1 Required Certificate Format

Certificates must be in **Base64** encoded **X.509** format.

6.3.2 Obtaining Certificates

Since certificates are simply files, they may be obtained through any means that you can obtain any other binary file, including

- an e-mail attachment
- via FTP
- downloading the certificate file from a Web server

Certificates have special meanings within SSL-aware Web browsers; such applications generally have special means to manage them. If your Web browser supports SSL security, it probably also provides a means to manage certificates. The instructions in the next few sections describe how to load and export certificates with Internet Explorer (the browser that is required for e*Gate). If you wish to perform these procedures using a different browser, see that browser's Help system.

Independent Certification Authorities

The following CAs are two of the most widely accepted sources for certificates:

- Verisign: **http://www.verisign.com/**
- Thawte Consulting: **http://www.thawte.com/**

Private Certification Authorities

There are a number of private certification authorities who provide both site and client certificates to a discrete group (for example, for exclusive use by a business's employees and clients). Private CA certificates can also be useful for permitting access to the issuing authority's site (for example, for a subscription service).

Obtaining CA Certificates From Secure Sites using Internet Explorer

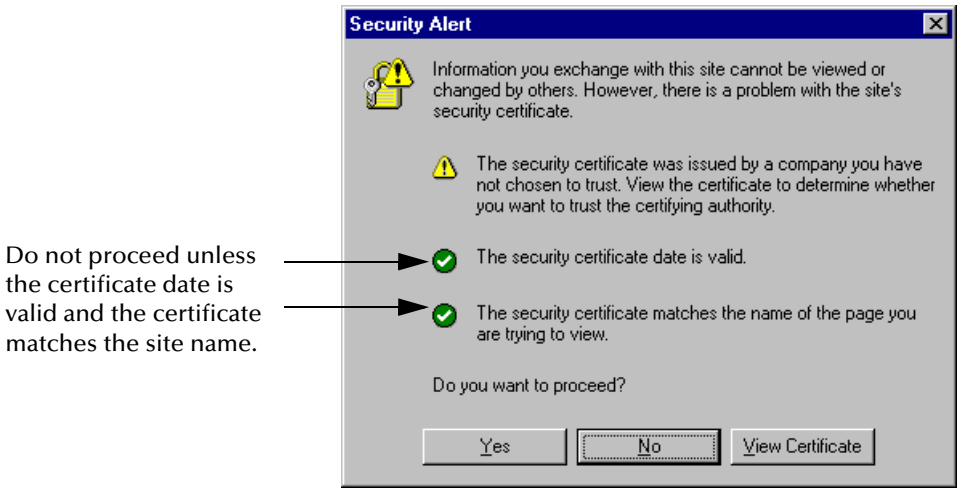
Microsoft's Internet Explorer warns you when you try to make a secure HTTP(S) connection to a site that is not within your Trusted Sites list. You can use this feature to obtain a certificate from the site.

To obtain a CA Certificate from a secure site

1 Using Internet Explorer, contact the secure site using an **HTTP(S):// URL** (for example, **HTTP(S)://www.securesite.com/**).

The browser displays an alert message (see **Figure 14 on page 81**).

Figure 14 Security Alert Display



2 You can do either or both of the following actions:

- Click **View Certificate** if you want to view the certificate details.
- To install the certificate, click **Yes**.

Once the certificate has been installed, you are able to view the secure site; you receive no further prompting or confirmation. After a certificate is installed, you can export it; see the next section for more information.

Exporting CA Certificates

This procedure exports a CA certificate from Internet Explorer to a file, which you can then commit to the e*Gate Registry.

To export a CA Certificate from within Internet Explorer

1 From the Tools menu, select **Internet Options**.

2 Select the **Content** tab.

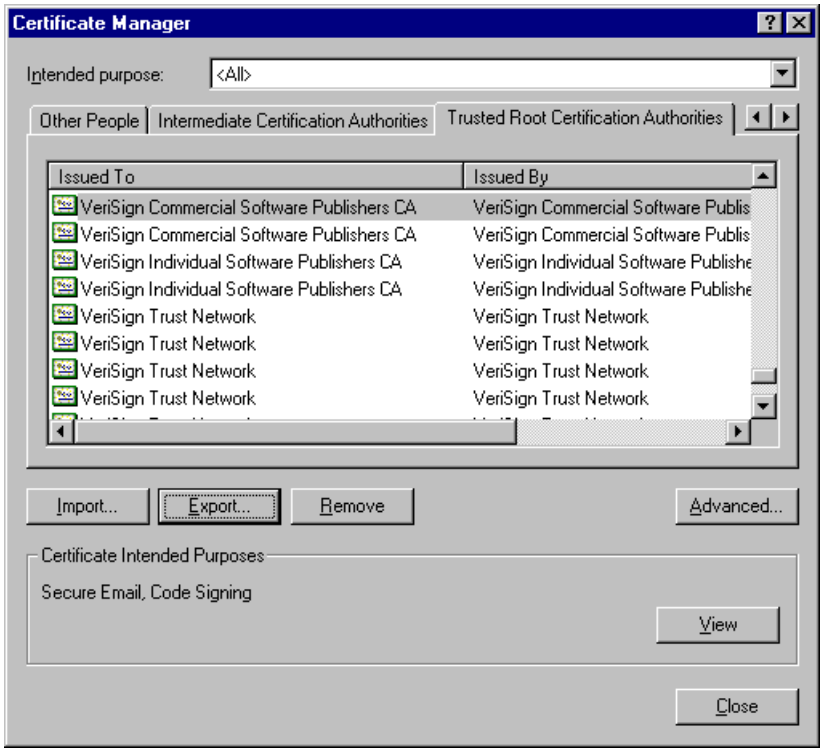
3 Click **Certificates**.

The Certificate Manager appears.

4 Select the **Trusted Root Certification Authorities** tab.

The selected tab lists the available certificates.

Figure 15 The Certificate Manager



5 Select the certificate you wish to export.

6 Click **Export**.

7 The Certificate Manager Export Wizard launches. Click **Next** to continue.

8 You are be asked to select a format. Select **Base64 encoded X.509 (.CER)**, then click **Next**.

9 You are prompted to enter a file name for the exported certificate. Enter a file name, then click **Next**.

10 You are prompted with a list of the choices you made while running the Wizard. Confirm that the choices are correct, then click **Finish**.

11 The Wizard reports success. Click **OK**.

Note: *The exported file is encoded in **Base64** format, so it is unreadable using a text editor such as Notepad.*

Working with Client Certificate/Key Pairs

Like CA certificates, client certificates are simply files, and can be retrieved using any method you use to retrieve any other file. However, not all client certificates can be managed as simply as CA certificates within Microsoft's Internet Explorer.

Fortunately, all of the other means to manipulate or retrieve files (for example, e-mail, FTP, download, or even a simple copy-and-paste operation) provide you with easy ways to obtain the client-certificate file itself.

Consult the issuing authority with any questions about key generation, certificate requests, or certificate management.

6.3.3 Importing Certificates to the e*Gate Registry

All certificates (both client and CA) must be committed to the e*Gate Registry before they are available for the HTTP(S) e*Way's use. You can import certificates to the e*Gate Registry using the **stcregutil** utility.

By default, CA certificates are stored in the repository directory:

```
pkicerts/trustedcas
```

We strongly recommend that you store CA certificates in this directory. The procedure below illustrates how to commit files to this directory.

We also recommend that you store client certificates in a **pkicerts/client/** directory (however, this directory is not created by default).

To import a certificate file

1. Log onto any system upon which the e*Gate GUIs or Participating Host components are installed.
2. Change to the directory where the certificate files are stored.
3. At the command prompt, type the following command:

```
stcregutil -rh RegHost -rs Schema -un User -up Passwd
    -fc pkicerts/trustedcas CertFile
```

Where the variables are defined as follows:

- ***RegHost*** and ***Schema*** are the names of the Registry Host and schema where the files are to be committed.
- ***User*** and ***Passwd*** are authentication information for an e*Gate user with sufficient privilege to commit the files.
- ***CertFile*** is the name of the certificate file itself.

A typical command line appears as follows:

```
stcregutil -rh My_host -rs Outbound_schema -un Administrator
    -up adminpass -fc pkicerts/trustedcas TradingPartner.cer
```

This command commits the file **TradingPartner.cer** to the Registry directory **pkicerts/trustedcas** on the Registry Host **My_host** within the schema named **Outbound_schema**. Validating the command is the Administrator user, with the password "adminpass."

Note: *You can also commit certificates to other directories within the Registry. Simply specify the desired directory after the* ***-fc*** *command flag. Optionally, you can commit files to the Registry using the e*Gate Schema Designer. See the Schema Designer's Help system for more information. See the* ***e*Gate Integrator System Administration and Operations Guide*** *for more information about the* ***stcregutil*** *utility and committing files to the e*Gate Registry.*

Chapter 7

HTTP(S) e*Way Implementation

This chapter explains how to implement the HTTP(S) e*Way Intelligent Adapter in a production environment.

7.1 HTTP(S) e*Way Implementation: Overview

The hyper-text transfer protocol (HTTP) sample in this implementation uses secure sockets layer (SSL) for data security, in other words, HTTP(S). To implement the HTTP(S) e*Way within the e*Gate Integrator system, you must do the following operations:

- Define Event Type Definitions (ETDs) to package the data being exchanged with the external system.

Note: *The HTTP(S) e*Way Extension (**stc_monkhttpnossl.dll**) is not thread-safe. It must only be used in an e*Way or a **single Collaboration** in a Business Object Broker (BOB).*

- In the e*Gate Schema Designer graphical user interface (GUI), do the following steps:
  - Define Collaboration Rules to process Event data.
  - Define any Intelligent Queues (IQs) to which Event data is published before sending it to the external system.
  - Define the e*Way component.
  - Within the e*Way component, configure Collaborations to apply the required Collaboration Rules.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's online Help or the **e*Gate Integrator User's Guide**.*

- Use the e*Way Editor to set the e*Way's configuration parameters (this procedure is explained in **Chapter 6**).
- Be sure that any other e*Gate components are configured as necessary to complete the schema.
- Test the schema and make any necessary corrections.

See **"Sample Configurations" on page 87** for examples of how the previous steps are combined to create a working implementation.

Note: *The delimiters for the configuration file must not appear within the URL string. The default delimiter set contains the equals sign (=). To modify this delimiter, open the configuration file, select* ***Options***, ***Config Delimiters***, *on the task bar, modify the value of delimiter 3 with a value that does not conflict with the search string.*

7.2 Creating Event Type Definitions from Form Data

You can use the ETD Editor to create or modify any necessary ETDs. However, if you wish to base ETDs upon existing HTML forms, you can automatically create these ETDs using the HTML Converter Build Tool.

The HTML Converter tool opens the HTML page, parses it for a <FORM> tag, and uses the structure within the form to build the ETD. Both POST and GET method types are supported. All <Input> types are supported except controls (such as submit and reset buttons) which do not send data to the server and are ignored.

Important: *If the form contains a link that is redirected to another Web page, you must save the source HTML code to a file on disk first, then use the local HTML file as the source for the HTML converter.*

There are two ways to launch the HTML Converter: from the command line and from the ETD Editor.

7.2.1 Creating Event Type Definitions using Command-line Utilities

To create an ETD using the HTML Converter command-line utility

From the command line, type the following on one line:

```
stc_form2ssc -rh registry_host -rs schema_name -un username
    -up password -html input_file -tf logfile output_file
```

Where:

- *registry_host* is the name of the computer on which the e*Gate Registry Host resides.
- *schema_name* is the name of the e*Gate schema you are creating. For requirements regarding schema names, see the Schema Designer's online Help system.
- *username* and *password* are the e*Gate administrator username and password, respectively.
- *input_file* is the HTML filename (including the path) or the URL to the HTML page.
- *logfile* is the name of a log file to capture warning and error messages. This argument is optional.
- *output_file* is the filename—including the path relative to the "eGate/client" directory—of the ETD file to be created. Specify the file extension— the converter will not supply the .**ssc** extension automatically.

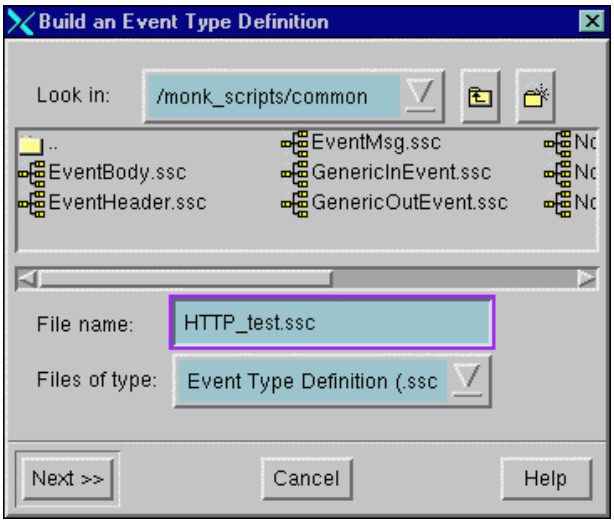
Note: *The **output_file** argument must be the last argument listed.*

7.2.2 Creating Event Type Definitions from the ETD Editor

To create an ETD using the HTML Converter from the ETD Editor

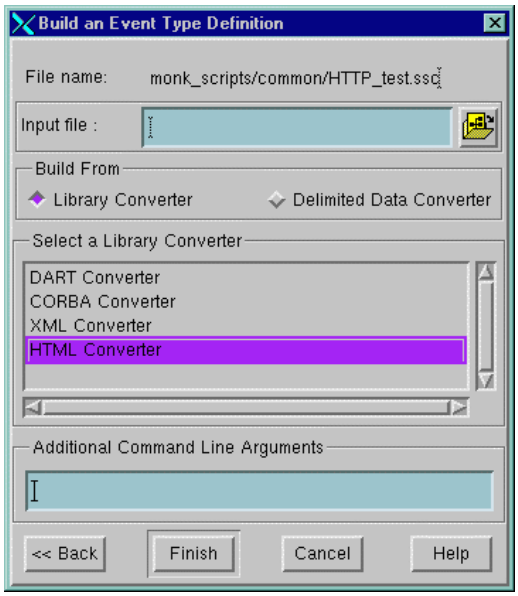
1. Launch the ETD Editor.
2. On the ETD Editor's Toolbar, click **Build**. The **Build an Event Type Definition** dialog box appears.

Figure 16 Build an Event Type Definition Dialog Box



3. In the **File name** field, type the name of the ETD file you wish to build. Do not specify any file extension. The Editor supplies the .**ssc** extension automatically.
4. Click **Next**. A new dialog box appears.

Figure 17 Build an Event Type Definition - HTML Converter



5 Leave the **Input file** field blank. The HTML Converter does not use this field.

6 Under **Build From**, select **Library Converter**.

7 Under **Select a Library Converter**, select **HTML Converter**.

8 Under **Additional Command Line Arguments**, type the following:

`-html` *input_file*

Where:

input_file is the HTML filename (including the path) or the URL to the HTML page.

9 Click **Finish**. The Build tool creates the ETD.

If the input HTML page contains more than one form, the Build tool will create multiple **.ssc** files, one for each form. The name of each file is the file name that was entered in step 3 in the previous procedure, plus an underscore and number (starting with zero), for example: **html_0.ssc**, **html_1.ssc**, and so on.

If your HTML page contained only a single form, the ETD Editor opens the resulting ETD file automatically at the conclusion of the conversion process. If multiple ETD files were created, you must open each file manually.

7.3 Sample Configurations

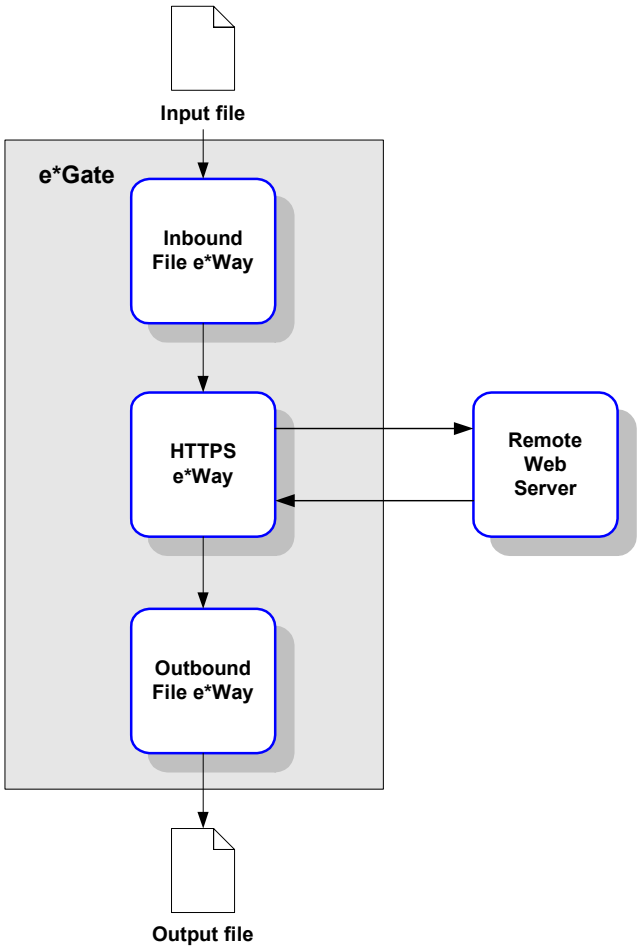
This section describes several sample implementations for the HTTP(S) e*Way.

7.3.1 Creating a Schema Using http-outgoing

This section demonstrates how to set up a basic schema using the **http-outgoing** function. In this sample, data is drawn from a text file using the file e*Way and sent to an external system using the HTTP(S) e*Way.

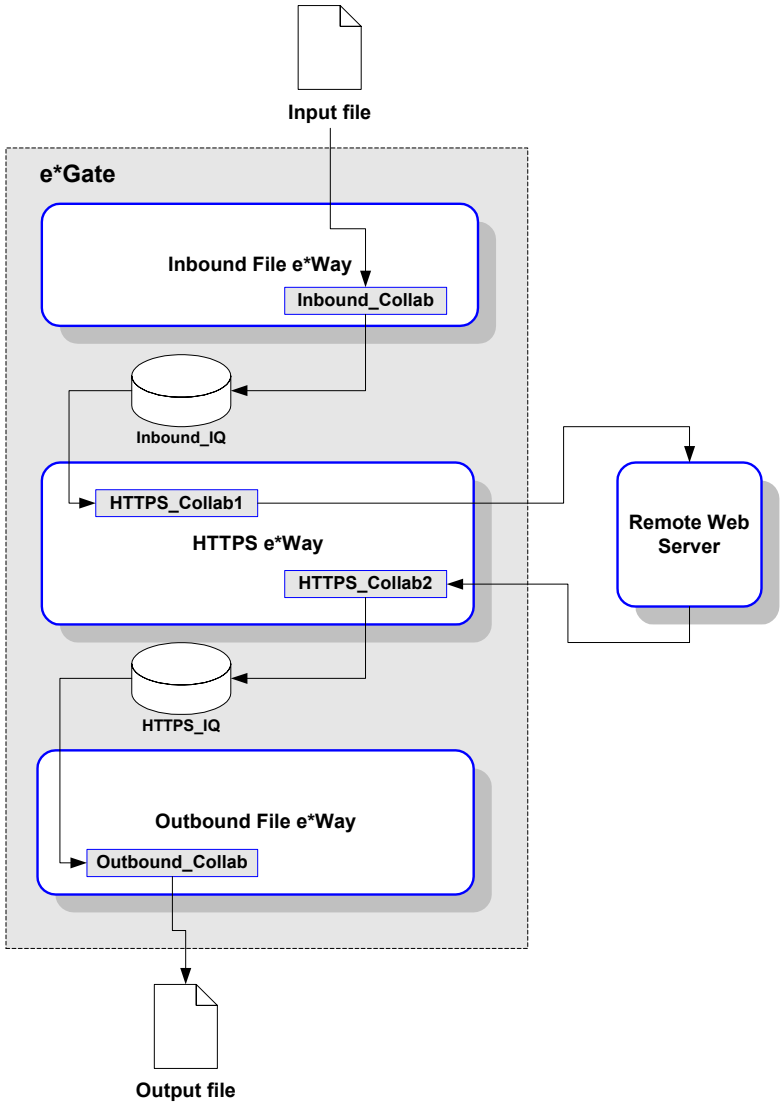
The data returned from the external system is received by the HTTP(S) e*Way, then forwarded to another file e*Way and stored in an output file on the local system (see Figure 18).

Figure 18 Sample Schema: Basic Architecture



This schema requires a number of components, as illustrated in **Figure 19 on page 89**.

Figure 19 Sample Schema: Component View



Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's online Help or the* ***e*Gate Integrator User's Guide****.*

1. Log into the e*Gate Schema Designer and click **New** to create a new schema. Name the schema "HTTP(S)_sample_1."

   The Schema Designer main screen appears.

2. If the Navigator's **Components** tab is not selected already, select it now.
3. Create an Event Type named "In."
4. Display the properties of the **In** Event Type. Then, use the **Find** button, navigate to the **common** folder to assign the file **GenericInEvent.ssc**.

5 Create a Collaboration Rule named "Passthrough_Data."

6 Edit the Properties of this Collaboration Rule as follows:

Service	Pass Through
Subscription	In (the Event Type defined in Step 1 above)
Publication	In (Event Type defined in Step 1 above)

7 Create two IQs, named "Inbound_IQ" and "HTTP(S)_IQ."

8 Create an e*Way named "Inbound."

9 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.

The next part of the procedure requires that you launch the e*Way editor and define the file-based e*Way's properties.

1 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

2 Using the e*Way Editor, make the following configuration settings:

Section	Parameter and Setting
General Settings	AllowIncoming: Yes AllowOutgoing: No
Poller(inbound) Settings	Polldirectory: C:\TEMP (or other "temporary" directory) Input File Mask: leave unchanged

3 Save the settings, promote to run time, and exit the e*Way Editor.

4 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create a Collaboration for the Inbound e*Way.

1 Open the **Inbound** e*Way and create a Collaboration named "Inbound_collab."

2 Set the Collaboration's properties as follows:

Collaboration Rule	Passthrough_Data
Subscriptions	Event: In Source: <External>.
Publications	Event: In Publish to: Inbound_IQ.

Now that the "inbound" e*Way is completely configured, you must create an outbound HTTP(S) e*Way.

1 Create a new e*Way component named "HTTP(S)_eway."

2 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewgenericmonk.exe**.

3 Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewhttp**.

4 Use the e*Way Editor to define the following parameters:

Section	Parameter and Settings
General Settings	Leave all settings unchanged
Communication Setup	Exchange Data Interval: 0 (zero) Zero Wait Between Successful Exchanges: No
Monk Configuration	Auxiliary Library Directories: monk_library/ewhttp Monk Environment Initialization File: monk_library/ewhttp/http-init.monk Startup Function: http-startup Process Outgoing Message Function: http-outgoing Exchange Data With External Function: http-exchange External Connection Establishment Function: http-connect External Connection Verification Function: http-verify External Connection Shutdown Function: http-shutdown Positive Acknowledgment Function: http-ack Negative Acknowledgment Function: http-nack The remaining parameters may be left blank for this sample.
HTTP Configuration	Timeout: 5000 User Name: enter an appropriate user name if necessary Encrypted Password: enter an appropriate password if necessary Agent: e*Gate HTTP(S) e*Way Content-type: Content-Type:application/x-www-form-urlencoded Accept-type: accept:text/* The remaining parameters may use the default values.
HTTP Proxy Configuration	Leave blank unless required
HTTP(S) Configuration	Leave blank to test basic HTTP functionality; if required, enter any necessary information to test HTTP(S) functionality

5 Save the settings, promote to runtime, and exit the e*Way Editor.

6 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create the Collaboration for the HTTP(S) e*Way.

1 Select the **HTTP(S)_eway** component and create a Collaboration named "HTTP(S)_collab1."

2 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: Inbound_collab
Publications	Event: In Publish to: <External>

3 Create a second Collaboration for the **HTTP(S)_eway**, naming it "HTTP(S)_collab2."

4 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: <External>
Publications	Event: In Publish to: HTTP(S)_IQ

Now create and configure the final e*Way component.

1 Create a new e*Way named "Outbound."

2 In its Properties Page, specify the executable file of "Outbound" as **stcewfile.exe**.

3 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.

4 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

5 Using the e*Way Editor, configuration the following settings:

Section	Parameter and setting
General Settings	AllowIncoming: No AllowOutgoing: Yes
Outbound (sender) Settings	Output directory: C:\TEMP (or other "temporary" directory) Output File Name: HTTP(S)_out.txt

6 Save the settings, promote to run time, and exit the e*Way Editor.

7 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

8 Create a Collaboration for the "Outbound" e*Way, naming it "outbound_collab."

9 Set the Collaboration's properties as follows:

Collaboration Rules:	Passthrough_Data
Subscriptions	Event: In Source: HTTP(S)_collab2
Publications	Event: In Publish to: <External>

The Schema Designer configuration is now complete. Now, you must create some test data which will be sent via HTTP(S) to external Web sites. The results of these requests will be saved to the output data file.

1 Use a text editor to create an input file. Create an Input File, using any ASCII text editor. The input must have the following format (the pipe symbol "|" delimits each field):

```
URL|POST or GET|data
```

An example follows, of the test data format:

```
HTTP(S)://info.somesite.com|GET|
HTTP(S)://finance.somesite.asp|POST|s=amd&d=v1
HTTP(S)://search.somesite.com/cgi-bin/
search|POST|search=Mars+missions
HTTP(S)://finance.somesite.com/q|GET|s=amd&d=v1
HTTP(S)://finance.somesite.com/q|GET|s=amd+&d=v4
HTTP(S)://finance.somesite.com/q|GET|s=amd&d=v1
```

Modify this sample according to the needs of your test sites.

2 Save the file as C:\TEMP\TESTDATA.FIN (if you specified a different input directory, please make the appropriate substitution).

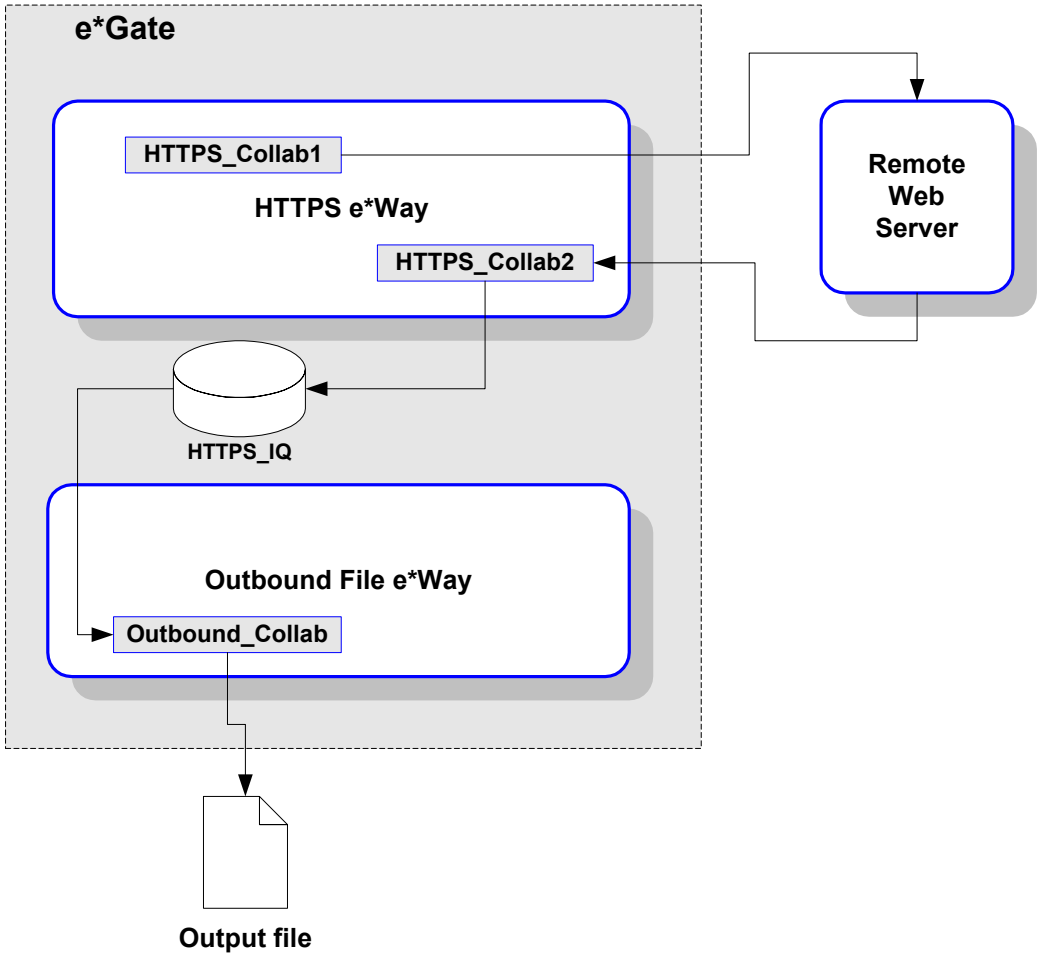
Run the sample schema. If the schema was configured properly and your connection to the test sites is good, you should find response data from your requests in the file **C:\TEMP\HTTP(S)_out.txt** (if you specified a different output directory, please make the appropriate substitution).

7.3.2 Creating a Schema Using http-exchange

This schema, which illustrates the use of the Monk function **http-exchange**, is simpler than the one illustrated in **"Creating a Schema Using http-outgoing" on page 88**.

Rather than using an inbound e*Way, the data to be sent to the external Web server is hard-coded into the HTTP(S) e*Way's configuration using the e*Way editor. Except for this change, the architecture is the same (see **Figure 20 on page 94**).

Figure 20 Sample http-exchange Schema



Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's Help system.*

To create a schema using http-exchange

1. Log into the e*Gate Schema Designer and select the New to create a new schema.
2. Enter the new schema name.
3. Create an Event Type named "In."
4. Display the properties of the **In** Event Type. Then, use the **Find** button to assign the file **GenericInEvent.ssc**.
5. Create a Collaboration Rule named "Passthrough_Data."

6 Edit the Properties of this Collaboration Rule as follows:

Service	Pass Through
Subscription	In (the Event Type defined in Step 1 above)
Publication	In (Event Type defined in Step 1 above)

7 Create an Intelligent Queue, named "HTTP(S)_IQ."

You must create an outbound HTTP(S) e*Way.

1 Create a new e*Way component named "HTTP(S)_eway."

2 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewgenericmonk.exe**.

3 Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewhttp**.

4 Use the e*Way Editor to define the following parameters:

Section	Parameter and Settings
General Settings	Leave all settings unchanged
Communication Setup	Exchange Data Interval: 10 (ten) Zero Wait Between Successful Exchanges: No
Monk Configuration	Auxiliary Library Directories: monk_library/ewhttp Monk Environment Initialization File: monk_library/ewhttp/http-init.monk Startup Function: http-startup Process Outgoing Message Function: http-outgoing Exchange Data With External Function: http-exchange External Connection Establishment Function: http-connect External Connection Verification Function: http-verify External Connection Shutdown Function: http-shutdown Positive Acknowledgment Function: http-ack Negative Acknowledgment Function: http-nack The remaining parameters may be left blank for this sample.

Section	Parameter and Settings (Continued)
HTTP Configuration	Request: GET Timeout: 5000 URL: enter an appropriate URL to contact. User Name: enter an appropriate user name if necessary Encrypted Password: enter an appropriate password if necessary Agent: e*Gate HTTP(S) e*Way Content-type: Content-Type:application/x-www-form-urlencoded Request-content: Leave this entry blank (because this is a sample using GET; fill in this field when using the POST method). Accept-type: accept:text/* The remaining parameters may use the default values.
HTTP Proxy Configuration	Leave blank unless required
HTTP(S) Configuration	Leave blank to test basic HTTP functionality; if required, enter any necessary information to test HTTP(S) functionality

5 Save the settings, promote to run time, and exit the e*Way Editor.

6 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create the Collaboration for the HTTP(S) e*Way.

1 Create a Collaboration for the **HTTP(S)_eway**, naming it "HTTP(S)_collab2."

2 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: <External>
Publications	Event: In Publish to: HTTP(S)_IQ

Now create and configure the final e*Way component.

1 Create a new e*Way named "Outbound."

2 In its Properties Page, specify the executable file of "Outbound" as **stcewfile.exe**.

3 Display the e*Way's properties. Then, use the **Find** button, navigate to the "**bin**" folder to assign the file **stcewfile.exe**.

4 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

5 Using the e*Way Editor, configuration the following settings:

Section	Parameter and setting
General Settings	AllowIncoming: No AllowOutgoing: Yes
Outbound (sender) Settings	Output directory: C:\TEMP (or other "temporary" directory) Output File Name: HTTP(S)_out.txt

6 Save the settings, promote to run time, and exit the e*Way Editor.

7 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

8 Create a Collaboration for the "Outbound" e*Way, naming it "outbound_collab."

9 Set the Collaboration's properties as follows:

Collaboration Rules:	Passthrough_Data
Subscriptions	Event: In Source: HTTP(S)_collab2
Publications	Event: In Publish to: <External>

The Schema Designer configuration is now complete. The results of these requests are saved to the output data file.

7.4 Sample Monk Scripts

The samples in this section can be run using the **stctrans** command-line utility. They do not require a complete e*Gate schema configuration to function, and are designed to illustrate the principles involved in creating your own custom Monk scripts. The library (dll) files to be loaded and the script to be tested must be in the load path (or, for simplicity's sake, may be placed in the current working directory).

Note: *See the Monk Developer's Reference for more information about the load path.*

The syntax of the **stctrans** utility is

```
stctrans monk_file.monk
```

Additional command-line flags are available; enter **stctrans -h** to display a list, or see the *e*Gate Integrator System Administration and Operations Guide* for more information.

The sample files may be created using any text editor. The samples use a generic "www.sitename.com" site name; before testing any script, replace the generic name with a working site name.

7.4.1 GET (Inbound) Example (HTTP_get)

The following script retrieves the URL **http://www.somesite.com** and displays the results:

```
;; Load HTTP extension DLL
(load-extension "stc_monkhttp.dll")

;; Create an HTTP session handle
(define hCon (http-acquire-provider "jdoe" "0E0102" "" "" 0))

;; Execute the HTTP GET method
(http-get hCon "http://www.somesite.com" 0 "accept:text/*")
(define pszData (http-get-result-data hCon)

;; Print the results
(display pszData)

;; Free HTTP session handle
(http-release-provider hCon)
(set! hCon 0)
```

Note: *Parameters could be passed by this script by appending them to the URL using the* ***application/x-www-form-urlencoded*** *format; for example,*

```
http://peterw?param1=16&param2=Lorne+Street
```

7.4.2 POST (Outbound) Example (HTTP_post)

The following script contains three examples: one posts to an ASP page, and the other two post to scripts at the specified URLs. The results are displayed.

```
;; Load HTTP extension DLL
(load-extension "stc_monkhttp.dll")

;; Create an HTTP session handle
(define hCon (http-acquire-provider "jdoe" "0E0102" "" "" 0))

;; Post to an Active Server Page (ASP) and print server reply
(define postCmd (http-post hCon "http://stingray/Project3/
Project3.asp" 0
"accept:text/*" "Content-Type: application/x-www-form-
urlencoded" "text1=doe"))
(if postCmd
     (begin
     (define postData (http-get-result-data hCon))
     (display postData)
     )
)

;; Post form data to a CGI script and print server reply
(define postCmd (http-post hCon "http://info.netscape.com/
home_search2.cgi"
0 "accept:text/*" "Content-Type: application/x-www-form-urlencoded"
"cp=Netscape&version=C&searchstring=Martin+Luther+King"))
(if postCmdHTTPS
     (begin
     (define postData (http-get-result-data hCon))
     (display postData)
     )
)
```

```
;; Post form data to a CGI script and print server reply
(define postCmd (http-post hCon "http://search.netscape.com/cgi-bin/
search"
0 "accept:text/*" "Content-Type: application/x-www-form-urlencoded"
"search=Mars+missions"))
(if postCmd
    (begin
    (define postData (http-get-result-data hCon))
    (display postData)
    )
)

;; Free HTTP session handle
(http-release-provider hCon)
(set! hCon 0)
```

7.4.3 Sample Input Data (AUTO_HTTP)

The sample in this section illustrates an input file for an inbound e*Way (change "somesite" to a valid site address).

Note: *When using an input file, it is necessary to modify the fields within the configuration file to match those within the input file, or to leave the fields blank. If a field in the configuration file, such as the* ***Request-content*** *parameter contains a string, and it does not appear within the input file, e*Gate will attempt to append the information. If within the input file, the delimiters are left empty the action within the configuration file will be used.*

The following input data is in the AUTO_HTTP sample schema and executes a POST or GET as specified and illustrates typical GET input data (boldface text) that could be passed to an HTTP(S) e*Way:

```
http://www.somesitea.com|GET|
http://www.somesitea.com|GET|
http://www.somesiteb.com|GET|
http://info.somesitec.com|GET|
http://finance.somesiteb.com/q|GET|s=amd&d=v1
http://finance.somesiteb.com/q|GET|s=stcs&d=v1
http://finance.somesiteb.com/q|GET|s=dell&d=v4
http://finance.somesiteb.com/q|GET|s=turf&d=b
http://www.somesited.com|GET|
http://www.somesitee.com|GET|
http://lc6.law5.hotmail.passport.com/cgi-bin/login|GET|
http://www.somesite-facts.com/
srchgrp.asp|POST|keywords=beef&stype=AND&group=ALL
http://www.msn.com|GET|
http://shop.infospace.com/cat1.htm?qvcid=539&qcat=416&nA=11|GET|
http://www.foxnews.com/video/main.sml|GET|
http://www.launch.com/music/welcome/pvn_musicvideos/?seti=1|GET|
http://www.trip.com/content/guidesandtools/0,1324,1-1,00.html|GET|
http://microsoft.com|GET|
http://www.datek.com|GET|
http://www.home.com|GET|
http://www.hotmail.com|GET|
http://www.stc.com|GET|
http://www.nutri-facts.com/
srchgrp.asp|POST|keywords=shrimp&stype=AND&group=ALL
http://www.yahoo.com|GET|
```

7.4.4 GET (Inbound) Example (HTTPS_get)

The following script retrieves the URL **HTTPS://www.*sitename*.com**:

```
;; function-list: http-init, http-establish-connection
;;load extension
(define http-init
 (lambda (  )
  (display "In http-init")
  (let ((return-value ""))
   (define except-base 3000)
   (define-exception except-abort (+ 0 except-base))
   (define-exception except-method (+ 1 except-base))
   (define-exception except-param (+ 2 except-base))
   (define-exception except-connect (+ 3 except-base))
   (define-exception except-transfer (+ 4 except-base))
   (define-exception except-local-op (+ 5 except-base))
   (define-exception except-rmt-op (+ 6 except-base))
   (define-exception except-rmt-list (+ 7 except-base))
   (if (load-extension "stc_monkhttp.dll")
    (begin
     "Successfully loaded DLL stc_monkhttp.dll"
    )
    (begin
     return-value
     "Failed to load DLL stc_monkhttp.dll"
    )
   )
   (define http-write-log
    (lambda ( x )
     (display x)
     #t
   ))
;;Specify whether using a Proxy server
(if (string=? HTTP_PROXY_CONFIGURATION_USE_PROXY_SERVER "NO")
    (begin
     (set! HTTP_PROXY_CONFIGURATION_USER_NAME "")
     (set! HTTP_PROXY_CONFIGURATION_ENCRYPTED_PASSWORD "")
     (set! HTTP_PROXY_CONFIGURATION_SERVER_ADDRESS "")
     (set! HTTP_PROXY_CONFIGURATION_PORT_NUMBER 0)
    )
    (begin
     (if (string? HTTP_PROXY_CONFIGURATION_PORT_NUMBER)
      (begin
       (set! HTTP_PROXY_CONFIGURATION_PORT_NUMBER (string-
>number HTTP_PROXY_CONFIGURATION_PORT_NUMBER))
      )
      (begin
      )
     )
    )
   )
   (if (string? HTTP_CONFIGURATION_TIMEOUT)
    (begin
     (set! HTTP_CONFIGURATION_TIMEOUT (string-
>number HTTP_CONFIGURATION_TIMEOUT))
    )
    (begin
     (set! HTTP_CONFIGURATION_TIMEOUT 1)
    )
   )
   (if (not (defined? HTTP_CONFIGURATION_ENCRYPTED_PASSWORD))
    (begin
     (set! HTTP_CONFIGURATION_ENCRYPTED_PASSWORD "")
```

```
     )
     (begin
     )
    )
    return-value
   )
))
;;establish connection
(define http-establish-connection
 (lambda (  )
;;Create an HTTPS session handle
(define hCon (http-acquire-
provider HTTP_CONFIGURATION_USER_NAME HTTP_CONFIGURATION_ENCRYPTED_PA
SSWORD HTTP_CONFIGURATION_AGENT HTTP_PROXY_CONFIGURATION_SERVER_ADDRE
SS 0))
  (if (string=? HTTP_PROXY_CONFIGURATION_USE_PROXY_SERVER "YES")
   (begin
;;Set Proxy properties
   (http-set-proxy-
properties hCon HTTP_PROXY_CONFIGURATION_SERVER_ADDRESS HTTP_PROXY_CO
NFIGURATION_PORT_NUMBER HTTP_PROXY_CONFIGURATION_USER_NAME HTTP_PROXY
_CONFIGURATION_ENCRYPTED_PASSWORD)
   )
   (begin
   )
  )
;;Load CA certificates
  (http-load-CA-certificates-
dir hCon HTTP_SSL_CONFIGURATION_TRUSTED_CA_CERTIFICATES_DIRECTORY)
  (if (string=? HTTP_SSL_CONFIGURATION_USE_CLIENT_CERTIFICATE_MAP "YE
S")
   (begin
;;Specify loacation of certificate map file.
  (define fRet (http-set-client-cert-from-
map hCon HTTP_CONFIGURATION_URL HTTP_SSL_CONFIGURATION_CLIENT_CERTIFI
CATE_MAP_FILE))
   )
   (begin
   )
  )
  hCon
))
;;Execute the HTTPS GET method
(define getCmd (http-get hCon "HTTPS://
www.somesite.com" 20000 "accept:text/*"))
(if getCmd
    (begin
    (define postData (http-post-get-result hCon))
    (display postData)
    )
)
```

Additional notes

In a typical HTTP(S) exchange, the client authenticates the server by obtaining a certificate from it, and verifies the certificate through a certificate authority (CA). This process occurs whenever a client requests a URL prefixed with **HTTPS**, as in **HTTPS://www.sitename.com/**. If the Monk script contains a URL specifying the HTTP(S) protocol, server authentication is performed by default using the specified CA in the Monk script.

Index

A

Accept-type **77**
Additional Path **67**
Agent **76**
AUTO_HTTP **26**
Auxiliary Library Directories **68**

C

Certificates **79–83**
  importing into the e*Gate Registry **83**
  obtaining **80**
  overview **49**
  required format **79**
Client Certificate Map File **79**
Communication Setup **57**
  Down Timeout **59**
  Exchange Event Interval **57**
  Resend Timeout **59**
  Start Exchange Data Schedule **58**
  Stop Exchange Data Schedule **59**
  Up Timeout **59**
  Zero Wait Between Successful Exchanges **60**
Components **8**
Configuration parameters **55–79**
Content-type **76**

D

Directories and files installed **14**
Down Timeout **59**

E

Encrypted Password **76**, **77**
example
  hard coded **26**
  inbound **25**
  outbound **26**
examples
  GET **25**, **98**
  httpnossl-outgoing **17**
  POST **26**, **98**
Exchange Data with External Function **70**
Exchange Event Interval **57**
export a CA Certificate from within Internet Explorer **81**
Exporting CA Certificates **81**
External Connection Establishment Function **71**
External Connection Shutdown Function **72**
External Connection Verification Function **72**

F

Forward External Errors **57**
functions
  http-acquire-provider **36**
  http-add-content-type-param **36**
  http-add-header **37**
  http-clear-content-type-param **38**
  http-clear-header **39**
  http-get **39**
  http-get-error-text **40**
  http-get-last-status **41**
  http-get-result-data **44**
  httpnossl-ack **29**
  httpnossl-connect **30**
  httpnossl-exchange **30**
  httpnossl-init **31**
  httpnossl-nack **31**
  httpnossl-notify **32**
  httpnossl-outgoing **32**
  httpnossl-shutdown **33**
  httpnossl-startup **34**
  httpnossl-verify **34**
  http-post **44**
  http-release-provider **46**
  http-set-proxy-properties **46**
  http-url-encode **48**

G

general operation, HTTP(S) e*Way **9**
General Settings **56**
  Forward External Errors **57**
  Journal File Name **56**
  Max Resends Per Event **56**
GET example **25**, **98**

H

hard coded example **26**
HTTP configuration **74**
  Encrypted Password **76**
  Request **74**
  Timeout **75**
  User name **75**

HTTP configurations
Accept-type 77
Agent 76
Content-type 76
Request-content 76
URL 75
HTTP functions
http-add-content-type-param 36
http-add-header 37
http-clear-content-type-param 38
http-clear-header 39
http-get-error-text 40
http-url-encode 48
HTTP Monk functions
http-acquire-provider 36
http-get 39
http-get-result-data 44
http-post 44
http-release-provider 46
http-set-proxy-properties 46
HTTP Proxy Configuration 77
Encrypted Password 77
Port Number 78
Server Address 78
User Name 77
HTTP Proxy configuration
Use Proxy Server 77
HTTP SSL Configuration 78
Client Certficate Map File 79
Trusted CA Certificates Directory 78
Use Client Certificate Map 78
http standard functions
httpnossl-ack 29
httpnossl-connect 30
httpnossl-exchange 30
httpnossl-nack 31
httpnossl-notify 32
httpnossl-outgoing 32
httpnossl-shutdown 33
httpnossl-startup 34
httpnossl-verify 34
HTTP_get 25
HTTP_post 26
http-acquire-provider 36
http-add-content-type-param 36
http-add-header 37
http-clear-content-type-param 38
http-clear-header 39
http-get 39
http-get-error-text 40
http-get-last-status 41
http-get-result-data 44
httpnossl-ack 29
httpnossl-connect 30
httpnossl-exchange 30
using 22
httpnossl-init 31
httpnossl-nack 31
httpnossl-notify 32
httpnossl-outgoing 17, 32
httpnossl-shutdown 33
httpnossl-startup 34
httpnossl-verify 34
http-post 44
http-release-provider 46
http-set-proxy-properties 46
http-standard functions
httpnossl-init 31
http-url-encode 48

I

importing certificates into the e*Gate Registry 83
inbound example 25
Independent Certification Authorities 80
installation
UNIX 13
Windows 12
Intended Reader 8

J

Journal File Name 56

M

Max Resends Per Event 56
Monk Configuration 60
Additional Path 67
Auxiliary Library Directories 68
Exchange Data with External Function 70
External Connection Establishment Function 71
External Connection Shutdown Function 72
External Connection Verification Function 72
Monk Environment Initialization 68
Negative Acknowledgment Function 73
Positive Acknowledgment Function 73
Process Outgoing Event Function 69
Shutdown Command Notification Function 74
Startup Function 69
Monk Environment Initialization File 68
monk functions
http-get-last-status 41

N

Negative Acknowledgment Function 73

O

Obtaining CA Certificates From Secure Sites using Internet Explorer **80**
Obtaining Certificates **80**
outbound example **26**

P

Parameters
- Additional Path **67**
- Auxiliary Library Directories **68**
- Down Timeout **59**
- Exchange Data with External Function **70**
- Exchange Event Interval **57**
- External Connection Establishment Function **71**
- External Connection Shutdown Function **72**
- External Connection Verification Function **72**
- Forward External Errors **57**
- general settings **56**
- Journal File Name **56**
- Max Resends Per Event **56**
- Monk Environment Initialization File **68**
- Negative Acknowledgment Function **73**
- Positive Acknowledgment Function **73**
- Process Outgoing Event Function **69**
- Resend Timeout **59**
- Shutdown Command Notification Function **74**
- Start Exchange Data Schedule **58**
- Startup Function **69**
- Stop Exchange Data Schedule **59**
- Up Timeout **59**
- Zero Wait Between Successful Exchanges **60**

Port Number **78**
Positive Acknowledgment Function **73**
POST example **26, 98**
Private Certification Authorities **80**
Process Outgoing Event Function **69**

R

Request **74**
Request-content **76**
Required certificate format **79**
Resend Timeout **59**

S

Secure Sockets Layer (SSL) overview **49**
security functions
- *See also* Certificates

Server Address **78**
Shutdown Command Notification Function **74**
Start Exchange Data Schedule **58**
Startup Function **69**
Stop Exchange Data Schedule **59**
system requirements **11**

T

Timeout **75**
Trusted CA Certificates Directory **78**

U

Up Timeout **59**
URL **75**
Use Client Certificate Map **78**
Use Proxy Server **77**
User Name **77**
User name **75**
using httpnossl-exchange **22**

W

Working with Certificates **79**
Working with Client Certificates **82**

Z

Zero Wait Between Successful Exchanges **60**