SeeBeyond™ eBusiness Integration Suite

HTTPS e*Way Intelligent Adapter User's Guide

Release 4.5.2

Monk-enabled

Version 20011019123757.

Contents

Chapter 4

Chapter 5

Chapter 1

Introduction

This document describes how to install and configure the HTTPS e*Way Intelligent Adapter.

1.1 Overview

The HTTPS e*Way provides a means to exchange data with a web (HTTP) server using the GET and POST methods.

Beyond simple data transfer with the HTTP protocol, the HTTPS e*Way offers the option of using HTTPS to provide secure data transport using the Secure Sockets Layer (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.

1.1.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have expert-level knowledge of Windows and/or UNIX operations and administration; to be thoroughly familiar with HTTPS certification, web servers, and Windows-style GUI operations.

1.1.2 Components

The following components comprise the HTTPS e*Way:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- A function-library file (**stc_monkhttp.dll**)
- Monk Event Type Definition Builder, a tool which builds a Monk Event Type Definition from a sample HTML page. See **"Creating Event Type Definitions from Form Data" on page 39**.

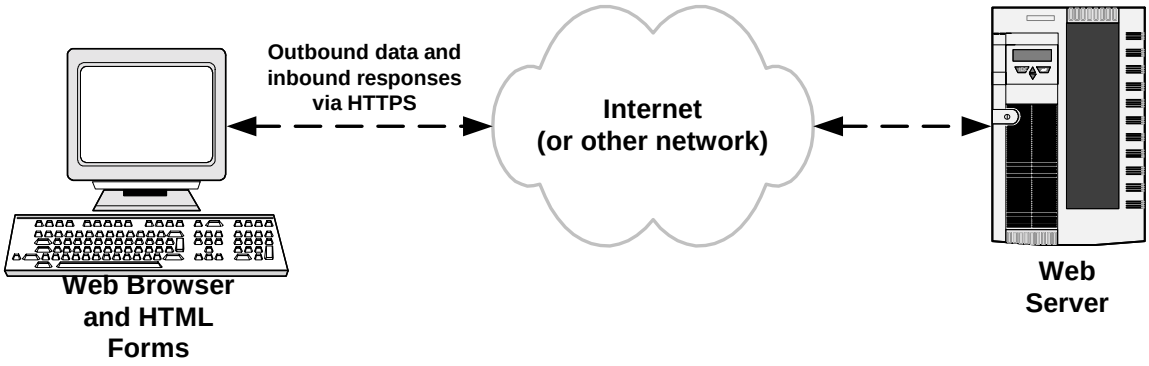
A complete list of installed files appears in **Table 1 on page 7**.

1.2 Operational Overview

1.2.1 Basic Operations

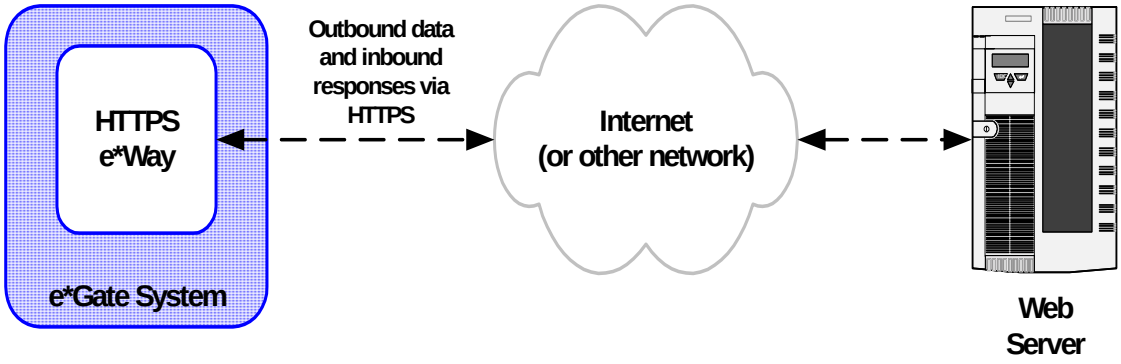
In a typical data exchange using HTTPS, a user sends requests to a web server using a web browser (for instance, when sending an order to an online shopping service using HTML forms).

Figure 1 HTTPS data exchange using a browser



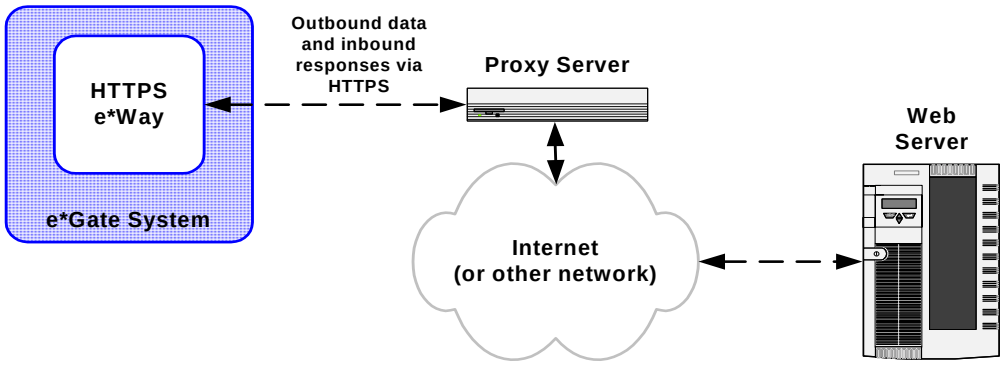
In an e*Gate implementation, the HTTPS e*Way exchanges data using the same HTTPS methods as a browser might.

Figure 2 HTTPS data exchange using the HTTPS e*Way



The HTTPS 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 firewall.

Figure 3 HTTPS data exchange through a firewall



1.2.2 Certificates and Security

The HTTPS 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 HTTPS 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.*

1.3 System Requirements

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

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

To use the HTTPS e*Way, you need the following:

- An e*Gate Participating Host, version 4.5 or higher. For AIX systems, you need an e*Gate Participating Host version 4.5.1.
- A TCP/IP network connection.

Chapter 2

Installation

This chapter describes how to install the HTTPS e*Way.

2.1 Windows NT or Windows 2000

2.1.1 Pre-installation

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

2.1.2 Installation Procedure

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

1. Log in as an Administrator on the workstation on which you want to install the e*Way Adapter.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use Windows Explorer to launch the file **setup.exe** on the CD-ROM drive.
4. The InstallShield setup application will launch. Follow the on-screen instructions to install the e*Way Adapter.

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

5. After the installation is complete, exit the install utility and launch the e*Gate Enterprise Manager.
6. In the Component editor, create a new e*Way Adapter.
7. Display the new e*Way Adapter's 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 will launch. Make any necessary changes, then save the configuration file.

13 You will 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 3**.

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

*For more information about configuring the e*Way or how to use the e*Way Editor, see the* ***Working with e*Ways*** *user guide.*

2.2 UNIX

2.2.1 Pre-installation

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

2.2.2 Installation Procedure

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

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

2 If necessary, mount the CD-ROM drive.

3 At the shell prompt, type

cd /cdrom

4 Start the installation script by typing:

setup.sh

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

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

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

6 After installation is complete, exit the installation utility and launch the e*Gate Enterprise Manager.

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 will launch. Make any necessary changes, then save the configuration file.

14 You will 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 3**.

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

*For more information about configuring e*Ways or how to use the e*Way Editor, see the* ***Working with e*Ways*** *user guide.*

2.3 Files/Directories Created by the Installation

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

Table 1 Files created by the installation

e*Gate Directory	File(s)
bin\	stcewgenericmonk.exe
bin\	stc_monkhttp.dll
configs\stcewgenericmonk\	stcewhttp.def
monk_library\	http.gui

Table 1 Files created by the installation

e*Gate Directory	File(s)
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
pkicerts\client\	certmap.txt
pkicerts\trustedcas\	GTECyberTrustGlobalRoot.cer MicrosoftRootAuthority.cer SecureServerCertificationAuthority.cer ThawtePremiumServerCA.cer ThawteServerCA.cer verisign_class3.cer

Chapter 3

Configuration

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

3.1 Introduction

This chapter describes the procedure for configuring a new HTTPS 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 Enterprise Manager's online help.

Before you can run the HTTPS e*Way, you must configure it using the e*Way Editor, which is accessed from the e*Gate Enterprise Manager GUI. The HTTPS e*Way package includes a default configuration file which you can modify using this editor.

3.2 e*Way Configuration Parameters

To change e*Way configuration parameters:

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

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *Working with e*Way* user's guide.

The e*Way's configuration parameters are organized into the following sections:

- General Settings
- Communication Setup
- Monk Configuration
- HTTP Configuration

- HTTP Proxy Configuration
- HTTPS Configuration

3.2.1 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file will be 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 will be journaled for the following conditions:

- When the number of resends is exceeded (see Max Resends Per Message below)
- When its receipt is due to an external error, but Forward External Errors is set to **No**. (See **"Forward External Errors" on page 11** for more information.)

Max Resends Per Message

Description

Specifies the maximum number of times the e*Way will attempt to resend a message to the external system after receiving an error. When this maximum number is reached, the message is considered "failed" and will be written to the journal file.

Required Values

An integer between 1 and 1,024. The default is 5.

Max Failed Messages

Description

Specifies the maximum number of failed messages that the e*Way will allow. When the specified number of failed messages is reached, the e*Way will shut down and exit.

Required Values

An integer between 1 and 1,024. 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 will be queued to the e*Way Adapter's configured queue. See **"Exchange Data with External Function" on page 24** for more information.

Required Values

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

See **"Schedule-driven data exchange functions" on page 19** for information about how the e*Way uses this function.

3.2.2 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 Enterprise Manager controls when the e*Way executable will run. The schedule you set within the parameters discussed in this section (using the e*Way Editor) determines when data will be 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 will be ignored and the e*Way will invoke the **Exchange Data with External Function** immediately.

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

See **"Down Timeout" on page 13** and **"Stop Exchange Data Schedule" on page 13** for more information about the data exchange schedule.

Zero Wait Between Successful Exchanges

Description

Selects whether to initiate data exchange 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 will immediately invoke the **Exchange Data with External** function if the previous exchange function returned data. If this parameter is set to **No**, the e*Way will always wait 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 24** for more information.

Start Exchange Data Schedule

Description

Establishes the schedule to invoke the e*Way Adapter'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 three of the following:

- **Exchange Data with External Function** on page 24
- **Positive Acknowledgment Function** on page 26
- **Negative Acknowledgment Function** on page 27

If you do not do so, the e*Way will terminate 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 (i.e., the 30th of every month would not include February).

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an ACK or 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 will be 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 24**, **"Exchange Data Interval" on page 11**, and **"Stop Exchange Data Schedule" on page 13** 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 (i.e., the 30th of every month would not include February).

Down Timeout

Description

Specifies the number of seconds that the e*Way will wait between calls to the **External Connection Establishment Function**. See **"External Connection Establishment Function" on page 25** 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 will wait between calls to the **External Connection Verification Function**. See **"External Connection Verification Function" on page 25** 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 will wait between attempts to resend a message to the external system, after receiving an error message.

Required Values

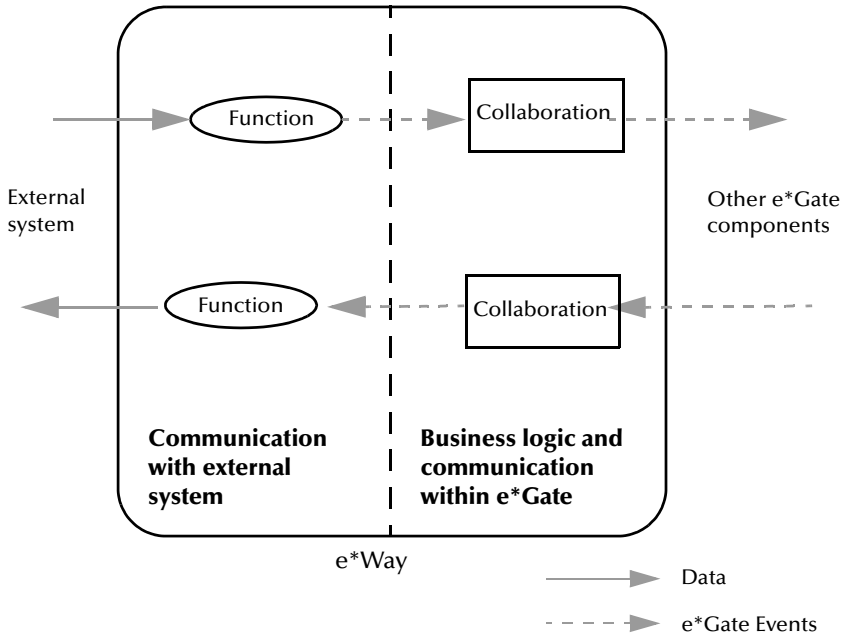
An integer between 1 and 86,400. The default is 10.

3.2.3 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.

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 4 below) 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 4 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 discussed 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

The Monk functions in the "communications half" of the e*Way fall into the following groups:

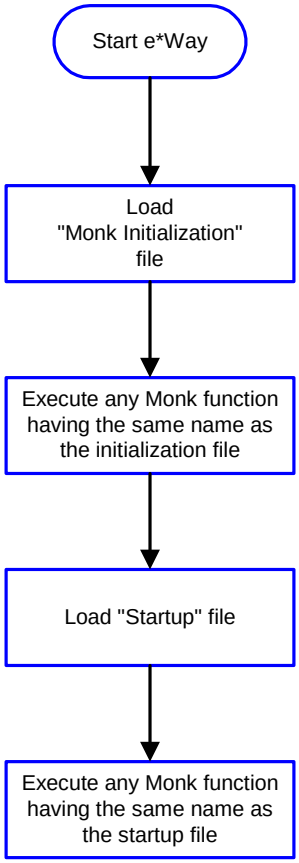
Type of Operation	Name
Initialization	**Startup Function** on page 23 (also see **Monk Environment Initialization File** on page 22)
Connection	**External Connection Establishment Function** on page 25 **External Connection Verification Function** on page 25 **External Connection Shutdown Function** on page 26
Schedule-driven data exchange	**Exchange Data with External Function** on page 24 **Positive Acknowledgment Function** on page 26 **Negative Acknowledgment Function** on page 27
Shutdown	**Shutdown Command Notification Function** on page 28
Event-driven data exchange	**Process Outgoing Message Function** on page 23

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

Initialization Functions

Figure 5 illustrates how the e*Way executes its initialization functions.

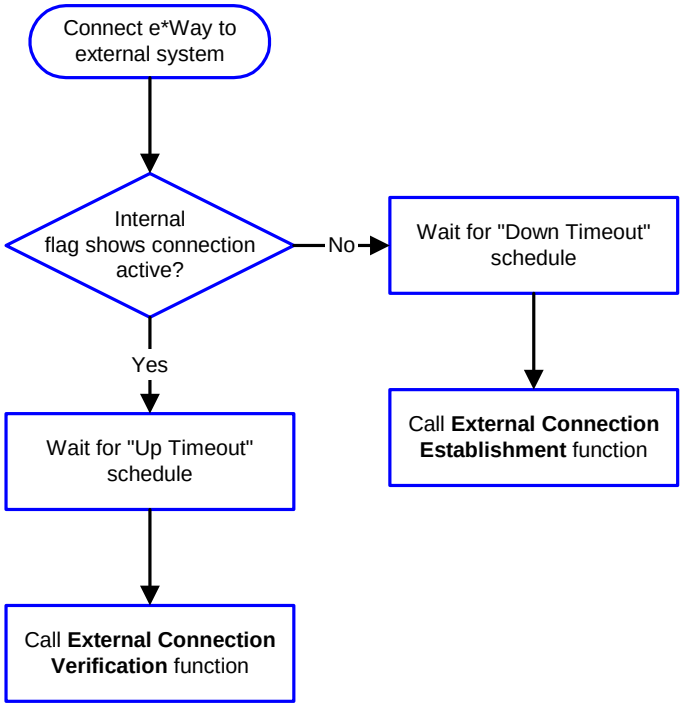
Figure 5 Initialization Functions



Connection Functions

Figure 6 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 6 Connection establishment and verification functions

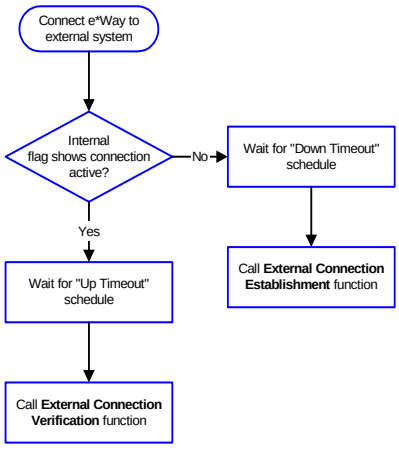


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 8 on page 19** *and* **Figure 10 on page 21** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **HTTP Standard Functions** *on page 63 and* **send-external-down** *on page 58 for more information.*

Figure 7 illustrates how the e*Way executes its "connection shutdown" function.

Figure 7 Connection shutdown function



Schedule-driven Data Exchange Functions

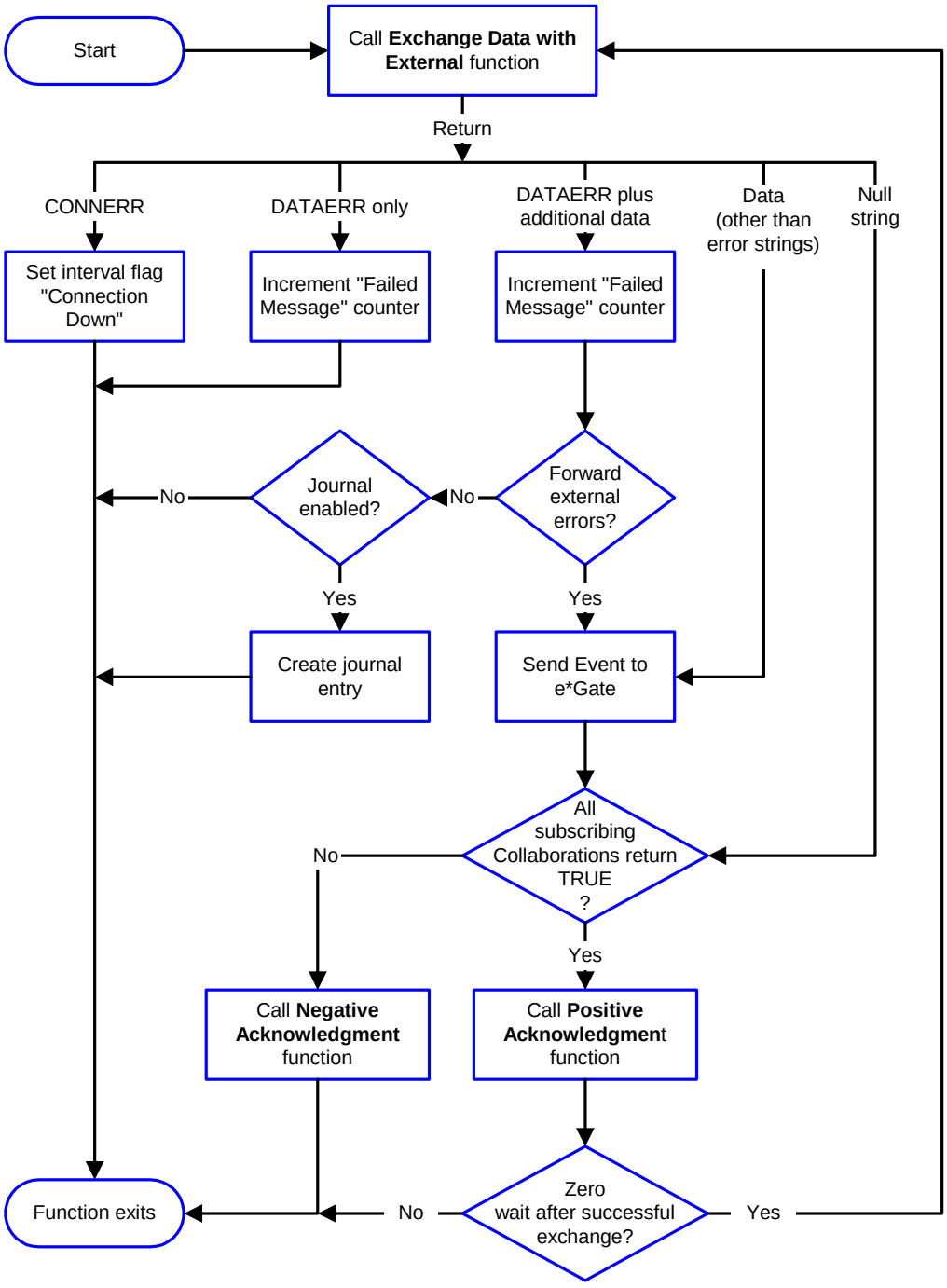
Figure 8 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgment Function** and **Negative Acknowledgment Function** are also called during this process.

"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.

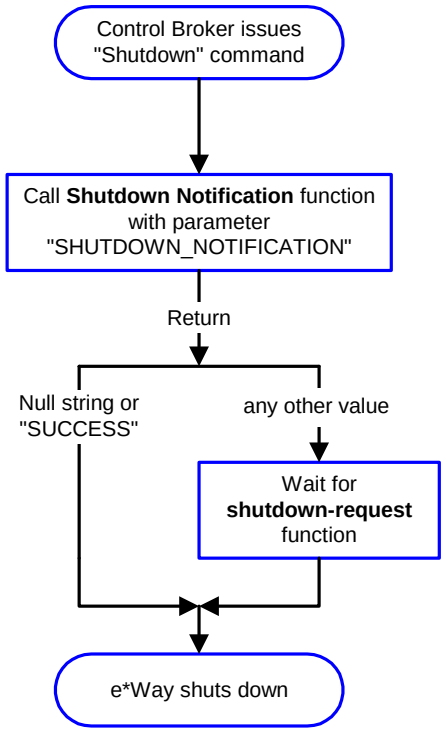
Figure 8 Schedule-driven data exchange functions



Shutdown Functions

Figure 9 illustrates how the e*Way implements the **shutdown request** function.

Figure 9 Shutdown functions



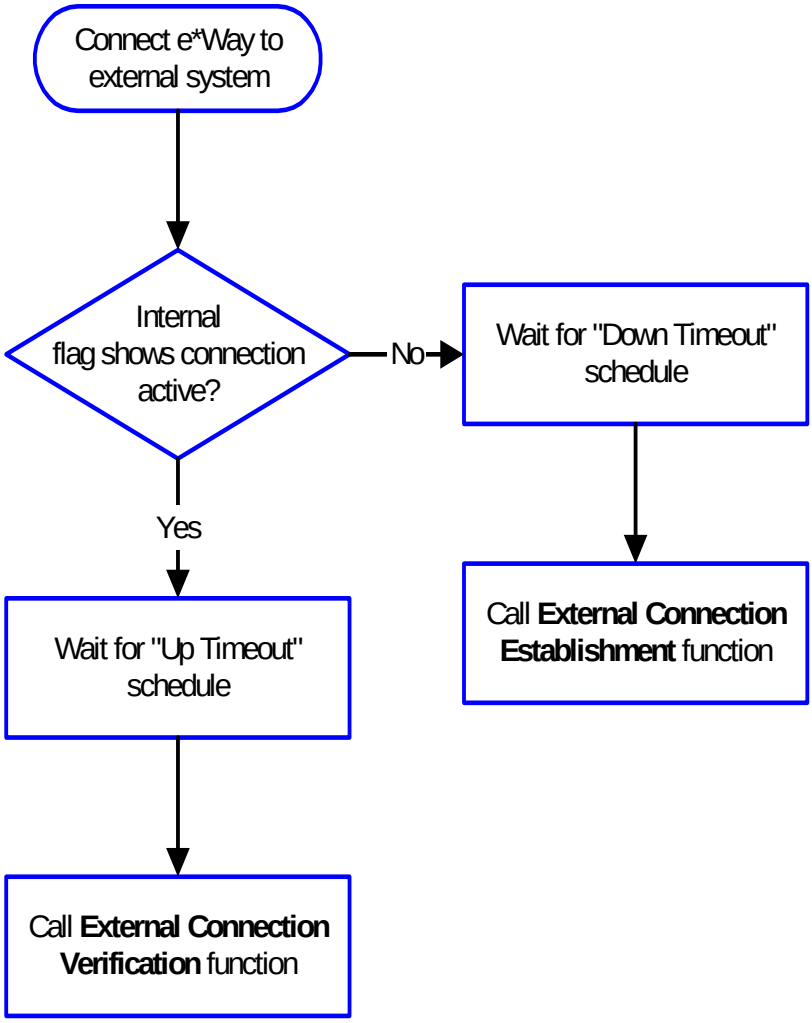
Event-driven Data Exchange Functions

Figure 10 on the next page 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 10 Event-driven data-exchange functions



How to Specify Function Names or File Names

Parameters that require the name of a Monk function will 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 will be searched before the default load path.

Required Values

A pathname, 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 will automatically be loaded into the e*Way Adapter's Monk environment.

Required Values

A pathname, 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 will be loaded after the auxiliary library directories are loaded. Typically, it is a good practice to initialize any global Monk variables that may be used by any other Monk Extension Scripts.

Required Values

A filename within the "load path", or filename plus path information (relative or absolute). If path information is specified, that path will be appended to the "load path." See **"Additional Path" on page 22** for more information about the "load path." (The default is **http-init.monk**. See **http-init** on page 67 for more information.)

Additional information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way will load this file and try 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 would attempt 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 5 on page 16**).

Startup Function

Description

Specifies a Monk function that the e*Way will load and invoke upon startup or whenever the e*Way Adapter'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 72 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 will be called after the e*Way loads the specified **Monk Environment Initialization file** and any files within the specified **Auxiliary Directories**.

The e*Way will load this file and try to invoke a function of the same base name as the file name (see **Figure 5 on page 16**). For example, for a file named **my-startup.monk**, the e*Way would attempt 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 70 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 Enterprise Manager). The function returns one of the following (see **Figure 10 on page 21** 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.
- If a string other than the following is returned, the e*Way will create an entry in the 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 56 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 66.)

Additional Information

The function accepts no input and must return a string (see **Figure 8 on page 19** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information will be 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 8 on page 19** 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 ACKed or NAKed (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 will immediately call the **Exchange Data with External** function again; otherwise, the e*Way will 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 61 for more information.)

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way will call 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 65 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 will call 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 will execute the **External Connection**

Establishment function in its place. (The default is **http-verify**. See **http-verify** on page 73 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 **Up Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Establishment** function (see above) 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 will call 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 71 for more information.)

Additional Information

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

This function will only be invoked when the e*Way receives a "suspend" command from a Control Broker. When the "suspend" command is received, the e*Way will invoke 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 will call when ***all*** the Collaborations to which the e*Way 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 64 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:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the Positive Acknowledgment function will be called again, with the same input data.
- Null string: The function completed 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 will call when the e*Way fails to process and queue 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 68 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:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the function will be called again.
- Null string: The function completed 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 will be called when the e*Way receives a "shut down" 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 Adapter, the e*Way will call this function with the string "SHUTDOWN_NOTIFICATION" passed as a parameter.

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

- A null string or "SUCCESS": Indicates that the shutdown can occur immediately.
- Any other string: Indicates that shutdown must be postponed. Once postponed, shutdown will not proceed until the Monk function **shutdown-request** is executed (See **shutdown-request** on page 60).

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.*

3.2.4 HTTP Configuration

This section defines the HTTP parameters used in the **http-acquire-provider** (See **http-acquire-provider** on page 77 for more information.) as well as the **GET** and **POST** calls. (See **Sample Configurations** on page 41 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 will await 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 should 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:

https://www.yourcompany.com:2080

or

https://www.yourcompnay.com/search2.cgi

Additional Information

If 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 or not 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 prior to receiving it.

User Name

Description

Specifies the username for authentication purposes necessary for connecting to the web server.

Required Values

A string containing any valid username. (See also **"Encrypted Password" on page 29**)

Additional Information

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

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

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 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 e*Way this was performed automatically. With this release it is necessary to call* **http-url-encode** *on page 90.*

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 will be ignored when the **GET** method is used.

The content will normally be 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/*”

3.2.5 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 will use the parameter values in this section to connect through a proxy server. Select **YES** if the e*Way should 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 username 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**.

3.2.6 HTTPS 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 CA certificates are located. These certificates will be used to verify a trust relationship between the user and the CA (Certification Authority).

Required Values

A relative pathname. The default is **pkicerts/trustedcas**.

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 & key (PEM)

For 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 will mean '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.

3.3 Certificates

3.3.1 Working with Certificates

Before the HTTPS 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, which the e*Way requires to establish a secure and trusted connection (see the **"Introduction" on page 1**" of this manual for more information about certificates).

3.3.2 Required Certificate Format

Certificates must be in Base64 encoded X.509 format.

3.3.3 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

Internet Explorer will warn you when you try to make a secure HTTPS 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 "https://" URL (for example, "https://www.securesite.com/").

The browser will display an alert (shown in the figure below).

Do not proceed unless the certificate date is valid and the certificate matches the site name.

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

Once the certificate has been installed, you will be able to view the secure site; you will 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 will export a CA certificate from Internet Explorer to a file, which you can then commit to the e*Gate Registry.

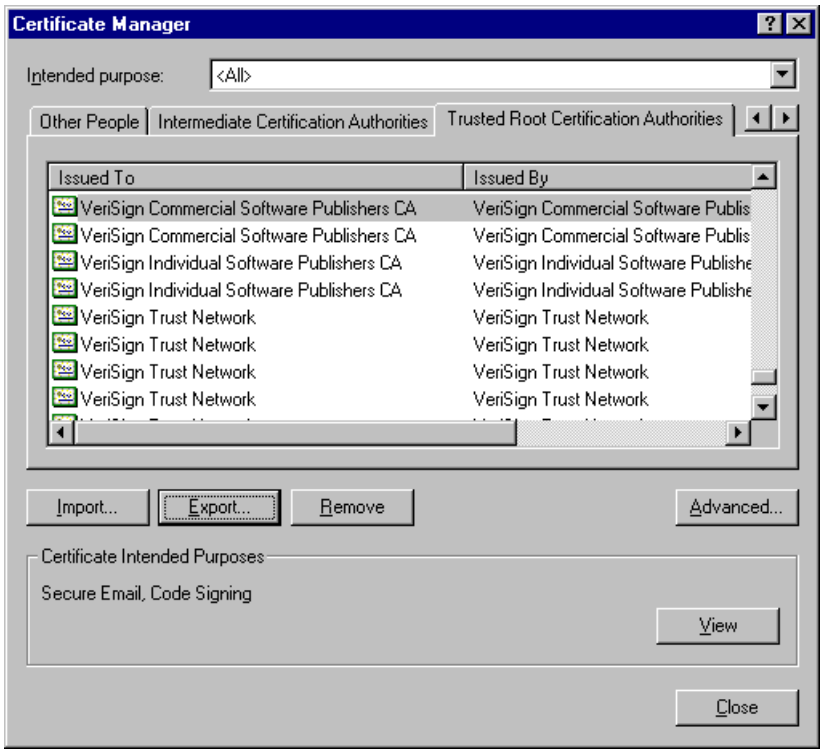
To export a CA Certificate from within Internet Explorer:

1. From the Tools menu, select **Internet Options**.
2. Select the **Content** tab.
3. Click **Certificates**.

   The Certificate Manager will appear.
4. Select the **Trusted Root Certification Authorities** tab.

   The selected tab will list the available certificates.

Figure 11 The Certificate Manager



5. Select the certificate you wish to export.
6. Click **Export**.
7. The Certificate Manager Export Wizard will launch. Click **Next** to continue.

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

9 You will be prompted to enter a file name for the exported certificate. Enter a filename, and then click **Next**.

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

11 The Wizard will report success. Click **OK**.

Note: *The exported file will be encoded in Base64 format, so it will be 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 Internet Explorer. Fortunately, all of the other means to manipulate or retrieve files (e-mail, FTP, download, or even a simple "copy and paste") provide you with easy means to obtain the client-certificate file itself. Consult the issuing authority with any questions regarding key generation, certificate requests, or certificate management.

3.3.4 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 HTTPS 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 in which the certificate files are stored.

3 At the command prompt, type the following:

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

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

A typical command line will look like the following:

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

This 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 Enterprise Manager. See the Enterprise Manager'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 4

Implementation

This chapter discusses how to implement the HTTPS e*Way.

4.1 Implementation Process: Overview

Note: *The HTTPS e*Way Extension (stc_monkhttp.dll) is not thread-safe. It must only be used in an e*Way or a SINGLE COLLABORTION in a BOB..*

To implement the HTTPS e*Way within an e*Gate system, do the following:

- Obtain any necessary CA and client certificates, and commit those certificates to the e*Gate Registry.
- Define Event Type Definitions (ETDs) to package the data being exchanged with the external system.
- In the e*Gate Enterprise Manager, do the following:
  - Define Collaboration Rules to process Event data.
  - Define any IQs to which Event data will be published prior to sending it to the external system.
  - Define the e*Way component (this procedure is discussed in **Chapter 2**).
  - Within the e*Way component, configure Collaborations to apply the required Collaboration Rules.
- Use the e*Way Editor to set the e*Way's configuration parameters.
- 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 41** for examples of how the above steps are combined to create a working implementation.

Note: *The delimiters for the configuration file must not appear within the URL string or the request-content 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.*

*For more information about creating or modifying any component within the e*Gate Enterprise Manager, see the Enterprise Manager's Help system.*

4.2 Creating Event Type Definitions from Form Data

You can use the ETD Editor to create or modify any necessary Event Type Definitions. 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 Event Type Definition. 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.

4.2.1 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 Enterprise Manager'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 Event Type Definition file to be created. Specify the file extension— the converter will not supply the .ssc extension automatically.

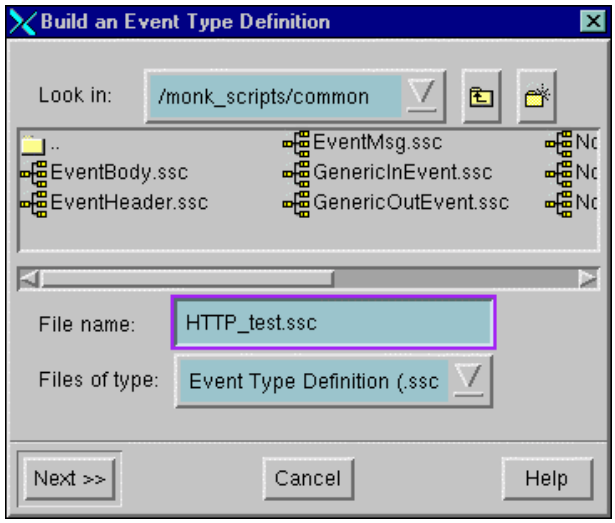
Note: *The **output_file** argument must be the last argument listed.*

4.2.2 Creating Event Type Definitions from the ETD Editor

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

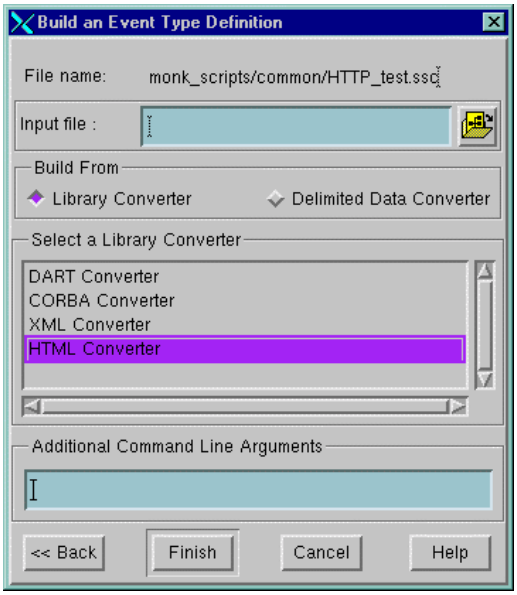
1. Launch the ETD Editor.
2. On the ETD Editor's Toolbar, click **Build**. The **Build an Event Type Definition** dialog box appears.

Figure 12 Build an Event Type Definition



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 will supply the .ssc extension automatically.
4. Click **Next**. A new dialog box appears.

Figure 13 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 will create 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 will be the file name that was entered in step 3 above, 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 will open the resulting ETD file automatically at the conclusion of the conversion process. If multiple ETD files were created, you must open each file manually.

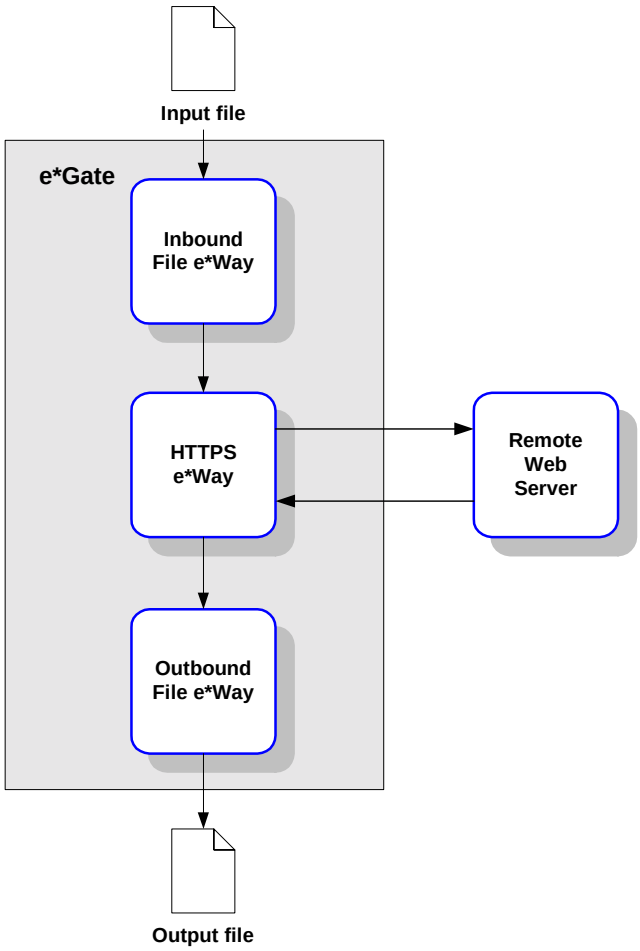
4.3 Sample Configurations

This section describes several sample implementations for the HTTPS e*Way.

4.3.1 Creating a Schema Using http-outgoing

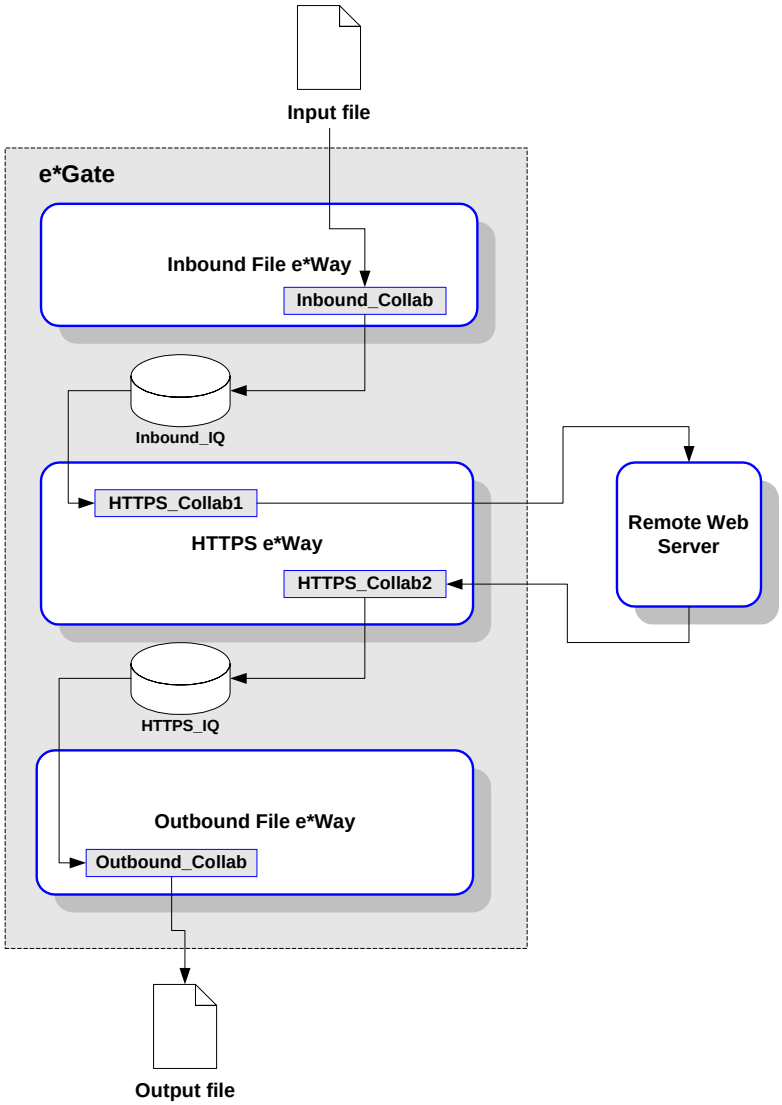
This section demonstrates how to set up a basic schema using the **http-outgoing** function. In this sample, data is drawn from a text file using the file e*Way and sent to an external system using the HTTPS e*Way. The data returned from the external system is received by the HTTPS e*Way, then forwarded to another file e*Way and stored in an output file on the local system (see **Figure 14 on page 42**).

Figure 14 Sample schema: basic architecture



This schema requires a number of components, as illustrated in **Figure 15 on page 43**.

Figure 15 Sample schema (component view)



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

1 Log into the e*Gate Enterprise Manager and click **New** to create a new schema. Name the schema "https_sample_1".

The Enterprise Manager 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 "HTTPS_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 Enterprise Manager'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 HTTPS e*Way.

1 Create a new e*Way component named "https_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 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
HTTPS Configuration	Leave blank to test basic HTTP functionality; if required, enter any necessary information to test HTTPS 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 Enterprise Manager's main window.

Next, create the Collaboration for the HTTPS e*Way.

1 Select the **https_eway** component and create a Collaboration named "https_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 **https_eway**, naming it "https_collab2".

4 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	Event: In Source: <External>
Publications	Event: In Publish to: HTTPS_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**: https_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 Enterprise Manager'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**: https_collab2
Publications	**Event**: In **Publish to**: <External>

The Enterprise Manager configuration is now complete. Now, you must create some test data which will be sent via HTTPS 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 (POST only)
```

The following is an example of the test data format. Modify according to the needs of your test sites.

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

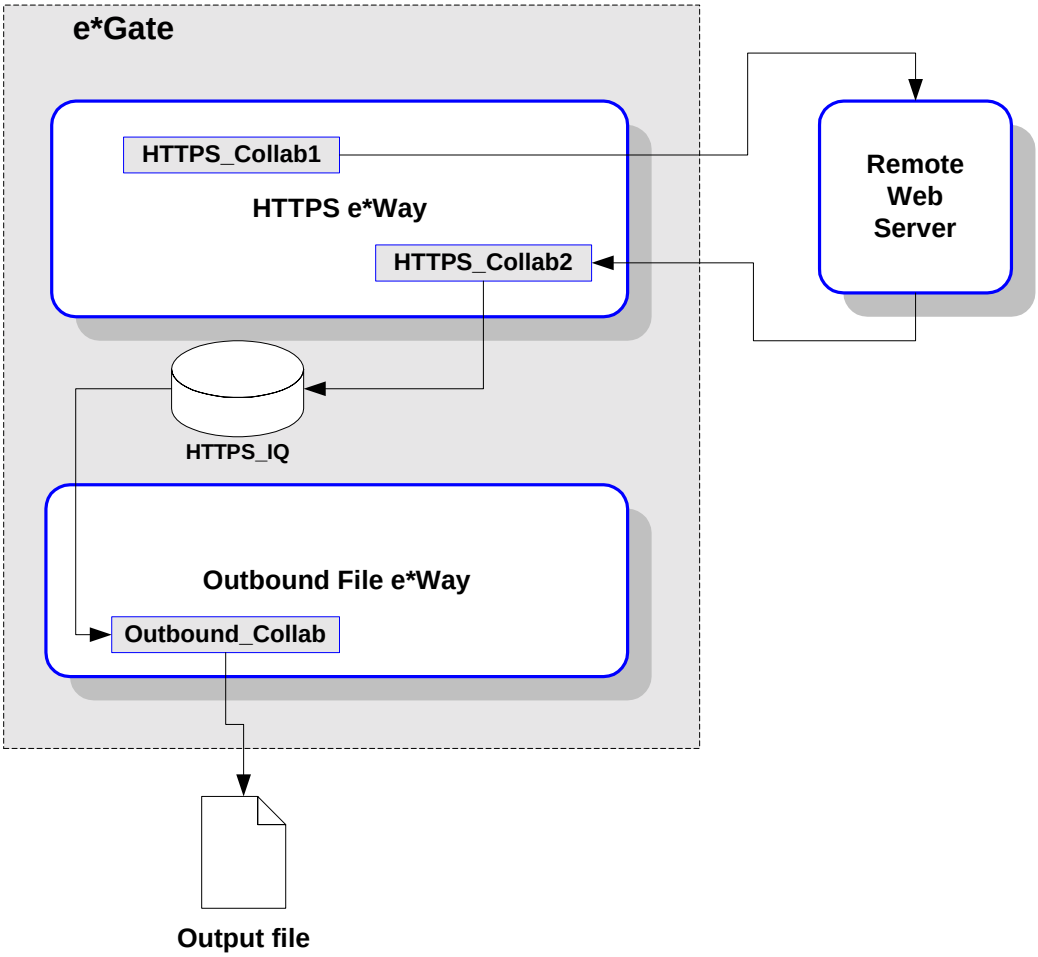
2 Save the file as C:\TEMP\TESTDATA.FIN (if you specified a different input directory, please make the appropriate substitution).

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\https_out.txt** (if you specified a different output directory, please make the appropriate substitution).

4.3.2 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 42**. Rather than using an inbound e*Way, the data to be sent to the external Web server is hard-coded into the HTTPS e*Way's configuration using the e*Way editor. Except for this change, the architecture is the same.

Figure 16 Sample http-exchange schema



Note: *For more information about creating or modifing any component within the e*Gate Enterprise Manager, see the Enterprise Manager's Help system.*

1. Log into the e*Gate Enterprise Manager 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 "HTTPS_IQ".

You must create an outbound HTTPS e*Way.

1 Create a new e*Way component named "https_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.
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 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
HTTPS Configuration	Leave blank to test basic HTTP functionality; if required, enter any necessary information to test HTTPS 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 Enterprise Manager's main window.

Next, create the Collaboration for the HTTPS e*Way.

1 Create a Collaboration for the **https_eway**, naming it "https_collab2".

2 Assign the following properties to the Collaboration:

Collaboration Rules	Passthrough_Data
Subscriptions	**Event**: In **Source**: <External>
Publications	**Event**: In **Publish to**: HTTPS_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**: https_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 Enterprise Manager'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**: https_collab2
Publications	**Event**: In **Publish to**: <External>

The Enterprise Manager configuration is now complete. The results of these requests will be saved to the output data file.

4.4 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). 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.

4.4.1 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
```

4.4.2 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)
```

4.4.3 Sample Input Data (AUTO_HTTP)

The sample below 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 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|
```

4.4.4 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_PASSWORD HTTP_CONFIGURATION_AGENT
HTTP_PROXY_CONFIGURATION_SERVER_ADDRESS 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_CONFIGURATION_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 "YES")
   (begin
;;Specify loacation of certificate map file.
  (define fRet (http-set-client-cert-from-map hCon HTTP_CONFIGURATION_URL
HTTP_SSL_CONFIGURATION_CLIENT_CERTIFICATE_MAP_FILE))
    )
    (begin
    )
  )
  hCon
))
;;Execute the HTTPS GET method
(define getCmd (http-get hCon "http://www.somesite.com" 20000 "accept:text/*"))
(if getCmd
      (begin
      (define postData (http-post-get-result hCon))
      (display postData)
      )
)
```

Additional notes

In a typical HTTPS exchange, the client authenticates the server by obtaining a certificate from it, and verifies the certificate through a Certificate Authority. This 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 "https" protocol, server authentication will be performed by default using the specified CA in the monk script.

Chapter 5

HTTPS e*Way Functions

The HTTPS e*Way functions fall into the following categories:

5.1 Basic Functions

The functions in this category control the e*Way's most basic operations.

The basic functions are

event-send-to-egate

Syntax

```
(event-send-to-egate string)
```

Description

event-send-to-egate sends an Event from the e*Way Adapter. If the external Collaboration(s) is successful in publishing the Event to the outbound queue, the function will return **#t**, otherwise **#f**.

Parameters

Name	Type	Description
string	string	The data to be sent to the e*Gate system.

Return Values

Boolean

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

Throws

None.

Additional Information

This function can be called by any e*Way function when it is necessary to send data to the e*Gate system in a blocking fashion.

get-logical-name

Syntax

```
(get-logical-name)
```

Description

get-logical-name returns the logical name of the e*Way Adapter.

Parameters

None.

Return Values

string
Returns the name of the e*Way (as defined by the Enterprise Manager).

Throws

None.

send-external-down

Syntax

```
(send-external-down)
```

Description

send-external down instructs the e*Way that the connection to the external system is down.

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

```
(send-external-up)
```

Description

send-external-up instructs the e*Way that the connection to the external system is up.

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown request requests the e*Way to perform the shutdown procedure when there is no outstanding incoming/outgoing event. When the e*Way is ready to act on the shutdown request, in invokes the **Shutdown Command Notification Function** (see **"Shutdown Command Notification Function" on page 28**). Once this function is called, the shutdown proceeds immediately.

Parameters

None.

Return Values

None.

Throws

None.

start-schedule

Syntax

```
(start-schedule)
```

Description

start-schedule requests that the e*Way execute the **Exchange Data with External** function specified within the e*Way Adapter's configuration file. Does not affect any defined schedules.

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

stop-schedule requests that the e*Way halt execution of the **Exchange Data with External** function specified within the e*Way Adapter's configuration file. Execution will be stopped when the e*Way concludes any open transaction. Does not affect any defined schedules, and does not halt the e*Way process itself.

Parameters

None.

Return Values

None.

Throws

None.

5.2 HTTP Standard Functions

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

The current suite of HTTP Monk standard functions are:

http-ack

Syntax

```
(http-ack message-string)
```

Description

http-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 will then be 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 will be called again.

Additional Information

See **"Positive Acknowledgment Function" on page 26** for more information.

http-connect

Syntax

```
(http-connect)
```

Description

http-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 25** for more information.

http-exchange

Syntax

```
(http-exchange)
```

Description

http-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 will be executed again with the same input Event.

Throws

None.

Additional Information

See **"Exchange Data with External Function" on page 24** for more information.

http-init

Syntax

```
(http-init)
```

Description

http-init begins the initialization process for the e*Way. This function loads the **stc_monkhttp.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 will shut 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 22** for more information.

http-nack

Syntax

```
(http-nack message-string)
```

Description

http-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 will be called again.

Throws

None.

Additional Information

See **"Negative Acknowledgment Function" on page 27** for more information.

http-notify

Syntax

```
(http-notify command)
```

Description

http-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 will pass 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 28** for more information.

http-outgoing

Syntax

```
(http-outgoing event-string)
```

Description

http-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 will be executed again 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 queue. See **event-send-to-egate** on page 56 for more information on this function.

Additional Information

See **"Process Outgoing Message Function" on page 23** for more information.

http-shutdown

Syntax

```
(http-shutdown shutdown)
```

Description

http-shutdown requests that the external connection shut down. A return value of "SUCCESS" indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a (**shutdown-request** on page 60) call 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 will pass 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 26** for more information.

http-startup

Syntax

```
(http-startup)
```

Description

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

Parameters

None.

Return Values

string
"FAILURE" causes shutdown of the e*Way. Any other return indicates success.

Throws

None.

Additional Information

This function should be used to initialize the external system before data exchange starts. Any additional variables may be defined here.

See **"Startup Function" on page 23** for more information.

http-verify

Syntax

```
(http-verify)
```

Description

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

Parameters

None.

Return Values

string
Returns "UP" if a connection is established. Any other value indicates the connection is not established.

Throws

None.

Additional Information

See **"External Connection Verification Function" on page 25** for more information.

5.3 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:

5.3.1 Rules for Encoding in the "x-www-form-urlencoded" Format

The following tables show representations for reserved characters, control characters, delimiters and symbols that are considered "unwise" to use. For more information on "x-www-form-urlencoded" rules, see **IETF RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax**. For more information, see **http-post** on page 87.

Table 2 Reserved Characters

Reserved Character	Escaped Character Representation
;	%3B
|	%7C
/	%2F
?	%3F
:	%3A
@	%40
&	%26
=	%3D
+	%2B
$	%24
,	%2C

Table 3 Control Characters

Control Character	Escaped Character Representation
ASCII 0	%00
ASCII 1	%01
ASCII 2	%02
ASCII 3	%03
ASCII 4	%04
ASCII 5	%05
ASCII 6	%06
ASCII 7	%07
ASCII 8	%08
ASCII 9	%09
ASCII 10	%0A
ASCII 11	%0B
ASCII 12	%0C
ASCII 13	%0D
ASCII 14	%0E
ASCII 15	%0F
ASCII 16	%10
ASCII 17	%11
ASCII 18	%12
ASCII 19	%13
ASCII 20	%14
ASCII 21	%15
ASCII 22	%16
ASCII 23	%17
ASCII 24	%18
ASCII 25	%19
ASCII 26	%1A
ASCII 27	%1B
ASCII 28	%1C
ASCII 29	%1D
ASCII 30	%1E
ASCII 31	%1F
ASCII 127	%7F
SPACE char	%20

Table 4 Delimiters

Delimiter Character	Escaped Character Value
<	%3C
>	%3E
#	%23
%	%25
“	%22

Table 5 Characters Not Recommended

Character	Escaped Character Value
{	%7B
}	%7D
|	%7C
\	%5C
^	%5E
[	%5B
]	%5D
‘	%60

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	a 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	an 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 error code returned by **http-get-last-status**.

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.
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 86 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)
      )
)
```

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.

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-post** 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.

Figure 17 Server Status Return Codes

Return Values	Description	Return CodeType
100	Continue	Information
101	Switching Protocols	Information
200	OK	Success
201	Create	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
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

Return Values	Description	Return CodeType
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
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

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** or **http-post** 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 a handle that is returned from **http-acquire-provider**. The return value is valid **only** when called after a FORM get as shown below (via **http-get** or **http-post**).*

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 will be 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, prior to calling **http-get-result-data**.

For more information regarding acceptable format types, see **Rules for Encoding in the “x-www-form-urlencoded” Format** on page 74.

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-release-provider

Syntax

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

Description

http-release-provider deallocates 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-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 without the port number. For example: "www.somesite.com"
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 HTTPS e*Way this was handled automatically. Currently, this function must be called in order to tranform the data string into urlencoded format.

5.4 HTTPS (Security) Functions

To access certain HTTP sites, it is necessary to set the certificate and private key information for the client server. The following functions provide this capability:

http-load-CA-certificates-dir on page 92

http-set-client-cert-from-map on page 94

http-load-CA-certificates-dir

Syntax

```
(http-load-CA-certificates-dir hCon directory)
```

Description

http-load-CA-certificates-dir specifies the directory from which to load the CA certificates.

Parameters

Name	Type	Description
hCon	opaque handle	The handle provided by **http-acquire-provider**.
directory	string	The path location where the certificates are stored.

Return Values

integer

Returns an integer specifying the number of certificate files loaded. If no certificate files are loaded, the return is zero (0).

Throws

None.

Additional Information

If **http-load-CA-certificates-dir** fails, no authentication can occur. A minimum of one valid certificate must exist in the specified location to authenticate against the server certificate.

The following table provides value definitions for the error codes returned that will be entered into the log file, provided that e*Way debug flags are set to "Monk Verbose".

Return Value	Description
0	ok
2	Error: unable to Get Issuer Certificate
3	Error: unable to get CRL
4	Error: unable to decrypt cert signature
5	Error: unable to decrypt CRL signature
6	Error: unable to decode issuer public key
7	Error: certificate signature failure
8	Error: CRL signature failure
9	Error: certificate not yet valid
10	Error: certificate has expired
11	Error: CRL not yet valid

Return Value	Description
12	Error: CRL has expired
13	Error: certificate not before field
14	Error: certificate not after field
15	Error: CRL last update field
16	Error: CRL next update field
17	Error: out of memory
18	Error: depth zero self signed certificate
19	Error: self signed certificate in chain
20	Error: unable to get issuer certificate locally
21	Error: unable to verify leaf signature
22	Error: certificate chain too long
23	Error: certificate revoked
50	Error: application verification

http-set-client-cert-from-map

Syntax

```
(http-set-client-cert-from-map hCon url mapfile)
```

Description

http-set-client-cert-from-map sets the logical location for the **Certificate Map File**.

Parameters

Name	Type	Description
hCon	opaque handle	The handle provided by **http-acquire-provider**.
url	string	The URL to which the certificate will be authenticated.
mapfile	string	The file name where the certificates are stored, including the relative path within the e*Gate Registry. For example: “pkicerts/client/certmap.txt”.

Return Values

Boolean

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

Throws

None.

Additional Information

See **“Client Certificate Map File” on page 33** for more information.

Index

A

B

C

D

E

F

G

H

I

J

M

N

O

P

R

S

T

U

W

Z