**SeeBeyond™ eBusiness Integration Suite**

# HTTP e*Way Intelligent Adapter User's Guide

*Release 4.5.2*

*Monk-enabled*

SEEBEYOND™

Version 20011019181247.

# Contents

Chapter 1

# Introduction

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

## 1.1 Overview

The HTTP e*Way is a Monk interface that enables the e*Gate system to exchange messages with an HTTP web server; that is, to upload (or post) messages to the web server and to download (get) data from it.

The HTTP Monk interface is a scriptable web client that offers a method of providing the data usually entered in a form in an html page to a file on a web server. This file may itself be a script that uses the data in an external application. The way in which the data is processed is entirely dependent on the external application and is of no concern to the HTTP Monk interface or the way in which it operates.

### 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 NT and/or UNIX operations and administration; to be thoroughly familiar with web servers and Windows-style GUI operations.

### 1.1.2 Components

The following components comprise the HTTP 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_nossl.dll**)
- HTML Converter, a tool which builds a Monk Event Type Definition from a sample HTML page. See **"Creating Event Type Definitions from Form Data" on page 31**.

A complete list of installed files appears in **Table 1 on page 5**.

## 1.2 System Requirements

The HTTP 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 11.0i
- HP-UX 11.0 and HP-UX 11.0i (Japanese)
- Compaq Tru64 Version 5.0A

To use the HTTP 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 HTTP e*Way.

## 2.1 Windows NT or Windows 2000

### 2.1.1 Pre-installation

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

### 2.1.2 Installation Procedure

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

1. Log in as an Administrator to the workstation on which you are installing the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's Autorun feature is enabled, the setup application launches automatically; skip ahead to step 4. Otherwise, use Windows Explorer to launch the file **setup.exe** on the CD-ROM drive.
4. The InstallShield setup application launches. Follow the installation instructions until you come to the **Please choose the product to install** dialog box.
5. Select **e*Gate Integrator**, then click **Next**.
6. Follow the on-screen instructions until you come to the second **Please choose the product to install** dialog box.
7. Clear the check boxes for all selections except **Add-ons**, and then click **Next**.
8. Follow the on-screen instructions until you come to the **Select Components** dialog box.
9. Highlight (but do not check) **e*Ways**, and then click the **Change** button. The **SelectSub-components** dialog box appears.

10 Select the **HTTP e*Way**. Click the continue button to return to the **Select Components** dialog box, then click **Next**.

11 Follow the rest of the on-screen instructions to install the HTTP e*Way. 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.**

***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.*

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

## 2.2 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 HTTP 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 the 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.***

## 2.3 Files/Directories Created by the Installation

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

**Table 1** Files created by the installation

| e*Gate Directory | File(s) |
|---|---|
| bin\ | stcewgenericmonk.exe<br>stc_monkfilesys.dll<br>stc_monkhttp_nossl.dll |
| configs\stcewgenericmonk\ | stcewhttpnossl.def |
| monk_library | httpnossl.gui |
| monk_library\ewhttpnossl\ | httpnossl-ack.monk<br>httpnossl-nack.monk<br>httpnossl-connect.monk<br>httpnossl-exchange.monk<br>httpnossl-init.monk<br>httpnossl-notify.monk<br>httpnossl-outgoing.monk<br>httpnossl-shutdown.monk<br>httpnossl-startup.monk<br>httpnossl-verify.monk |

Chapter 3

# Configuration

This chapter describes the procedure for configuring a new e*Way. You can also edit an existing e*Way and rename an e*Way. Procedures for creating and editing e*Gate components are provided in the Enterprise Manager's online help.

## 3.1 Introduction

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

## 3.2 e*Way Configuration Parameters

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

**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 *e*Gate Integrator 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

## 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 8** 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's configured queue. See **"Exchange Data with External Function" on page 21** for more information.

#### Required Values

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

## 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 that you set using the e*Way's properties in the Enterprise Manager controls when the e*Way executable will run. The schedule that you set within the e*Way Editor determines when data will be exchanged. Be sure that 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 10** and **"Stop Exchange Data Schedule" on page 9** 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 a 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 21** for more information.

## Start Exchange Data Schedule

### Description

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

### Required Values

One of the following:

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

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

- **Exchange Data with External Function** on page 21
- **Positive Acknowledgment Function** on page 23
- **Negative Acknowledgment Function** on page 24

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 21**, **"Exchange Data Interval" on page 8**, and **"Stop Exchange Data Schedule" on page 9** 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 22** 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 22** 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

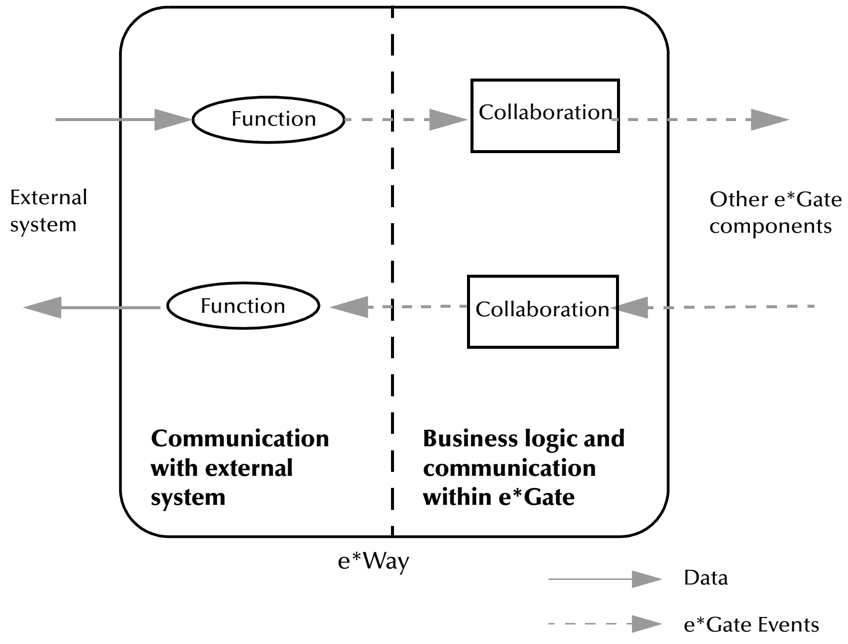
An integer between 1 and 86,400. The default is **10**.

## 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 1 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 1** 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:

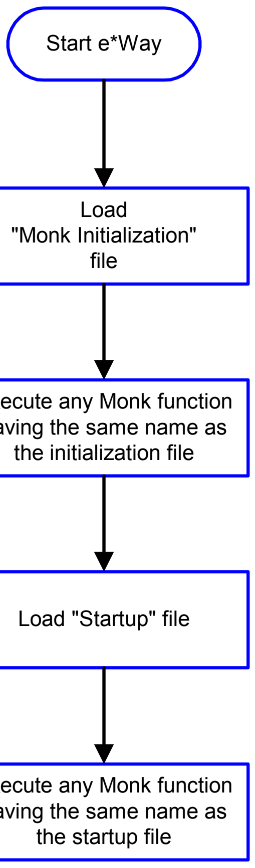
| Type of Operation | Name |
|---|---|
| Initialization | **Startup Function** on page 20 (also see **Monk Environment Initialization File** on page 19) |
| Connection | **External Connection Establishment Function** on page 22<br>**External Connection Verification Function** on page 22<br>**External Connection Shutdown Function** on page 23 |
| Schedule-driven data exchange | **Exchange Data with External Function** on page 21<br>**Positive Acknowledgment Function** on page 23<br>**Negative Acknowledgment Function** on page 24 |
| Shutdown | **Shutdown Command Notification Function** on page 25 |
| Event-driven data exchange | **Process Outgoing Message Function** on page 20 |

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

### Initialization Functions

Figure 2 illustrates how the e*Way executes its initialization functions.

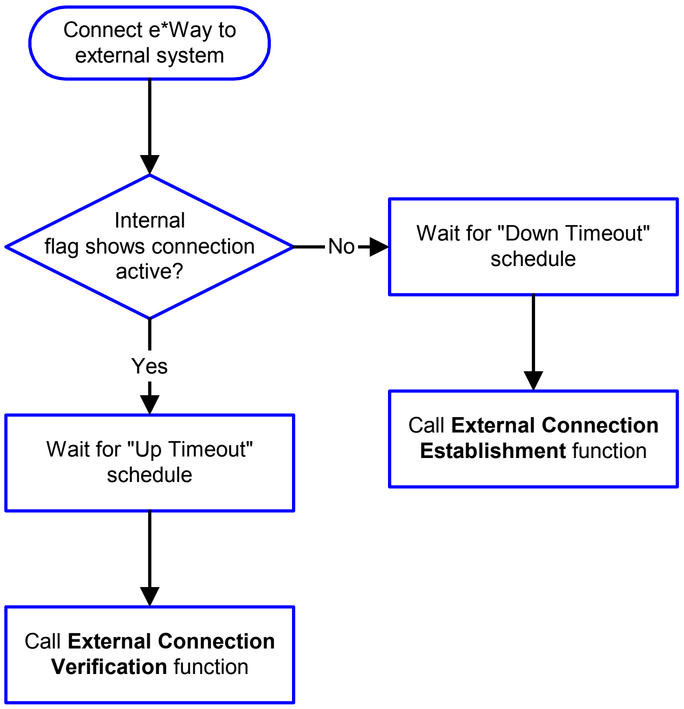
**Figure 2** Initialization Functions


Start e*Way
Load "Monk Initialization" file
Execute any Monk function having the same name as the initialization file
Load "Startup" file
Execute any Monk function having the same name as the startup file

### Connection Functions

Figure 3 illustrates how the e*Way executes the connection establishment and verification functions.

**Figure 3** Connection establishment and verification functions

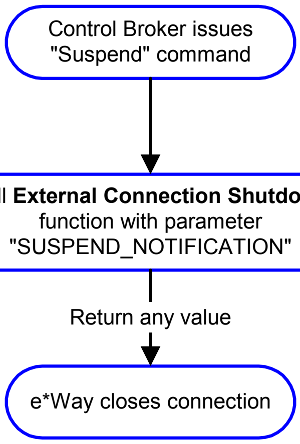

Connect e*Way to external system
Internal flag shows connection active?
No
Wait for "Down Timeout" schedule
Call **External Connection Establishment** function
Yes
Wait for "Up Timeout" schedule
Call **External Connection Verification** function

**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 5 on page 16** *and* **Figure 7 on page 18** *for examples of how different functions use this flag.*

*User functions can manually set this flag using Monk functions. See* **send-external-up** *on page 50 and* **send-external-down** *on page 49 for more information.*

Figure 4 illustrates how the e*Way executes its "connection shutdown" function.

**Figure 4** Connection shutdown function


Control Broker issues "Suspend" command
Call **External Connection Shutdown** function with parameter "SUSPEND_NOTIFICATION"
Return any value
e*Way closes connection

### Schedule-driven Data Exchange Functions

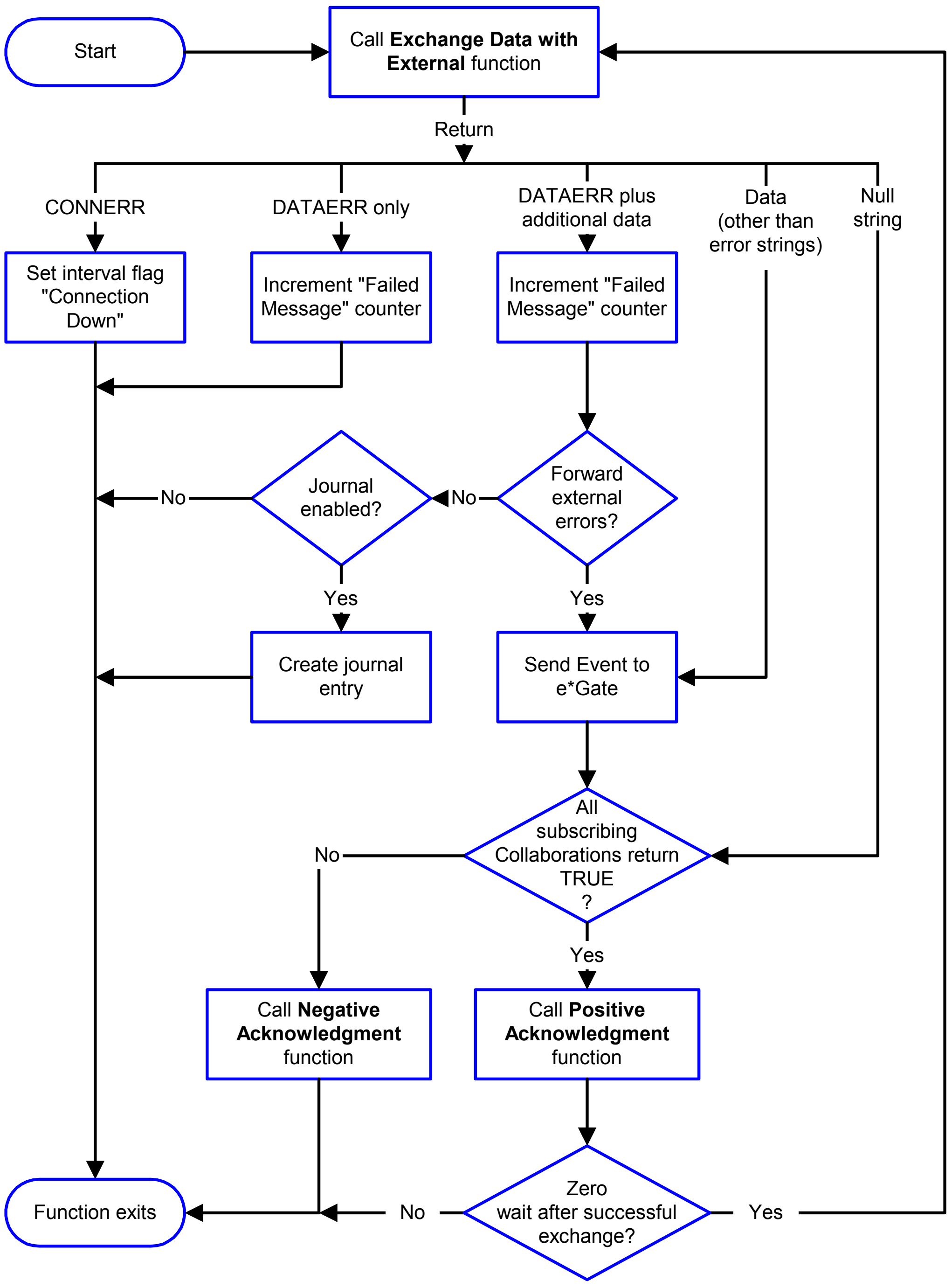
Figure 5 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

"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.

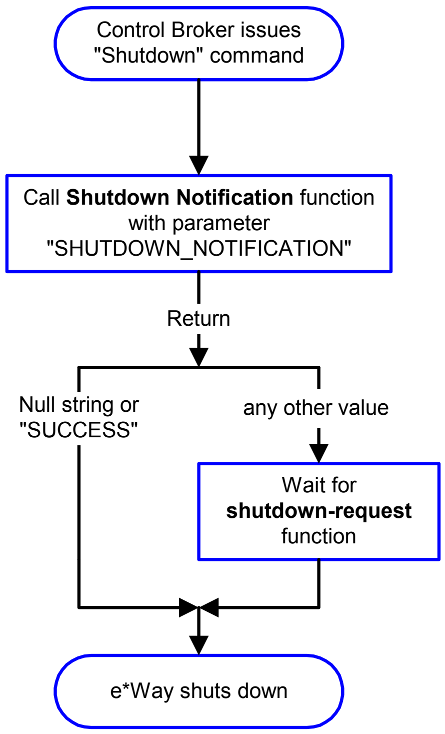
**Figure 5** Schedule-driven data exchange functions


Start
Call Exchange Data with External function
Return
CONNERR
DATAERR only
DATAERR plus additional data
Data (other than error strings)
Null string
Set interval flag "Connection Down"
Increment "Failed Message" counter
Increment "Failed Message" counter
Journal enabled?
No
No
Forward external errors?
Yes
Yes
Create journal entry
Send Event to e*Gate
All subscribing Collaborations return TRUE ?
No
Yes
Call Negative Acknowledgment function
Call Positive Acknowledgment function
Function exits
No
Zero wait after successful exchange?
Yes

## Shutdown Functions

Figure 6 illustrates how the e*Way implements the shutdown request function.

**Figure 6** Shutdown functions


Control Broker issues "Shutdown" command
Call **Shutdown Notification** function with parameter "SHUTDOWN_NOTIFICATION"
Return
Null string or "SUCCESS"
any other value
Wait for **shutdown-request** function
e*Way shuts down

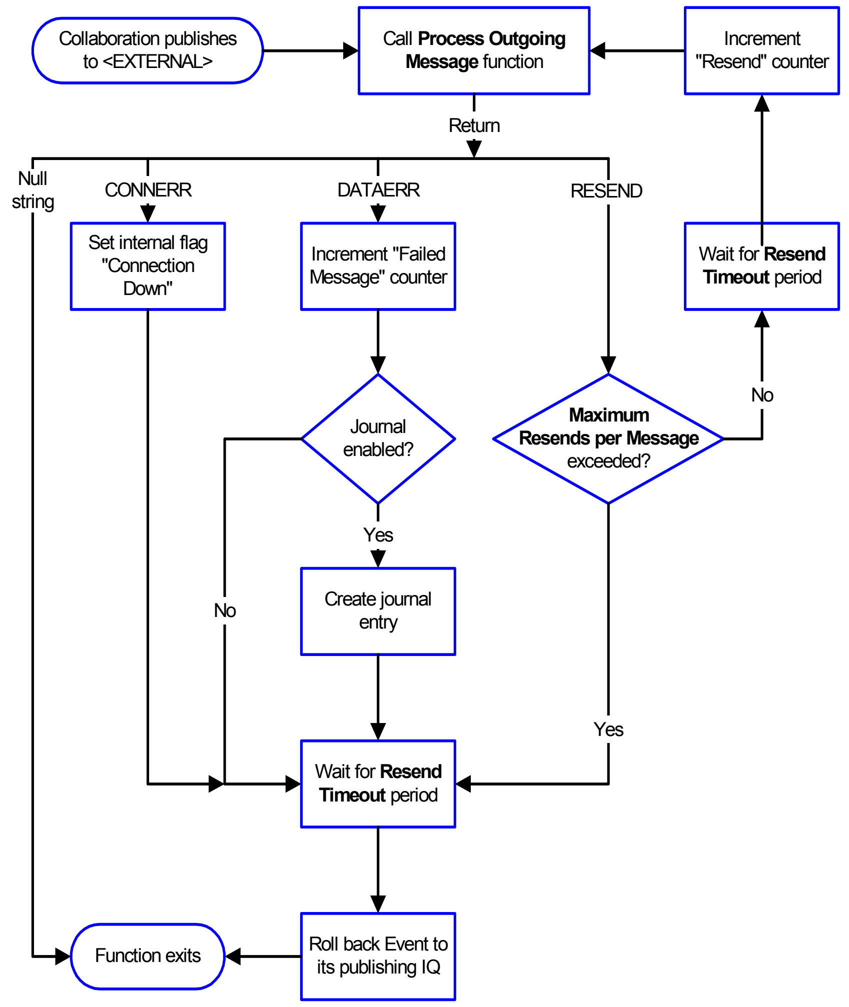
### Event-driven Data Exchange Functions

Figure 7 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 7** Event-driven data-exchange functions


Collaboration publishes to <EXTERNAL>
Call **Process Outgoing Message** function
Increment "Resend" counter
Return
Null string
CONNERR
DATAERR
RESEND
Set internal flag "Connection Down"
Increment "Failed Message" counter
Wait for **Resend Timeout** period
Journal enabled?
**Maximum Resends per Message** exceeded?
No
Yes
Create journal entry
No
Yes
Wait for **Resend Timeout** period
Function exits
Roll back Event to its publishing IQ

## 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's Monk environment.

### Required Values

A pathname, or a series of paths separated by semicolons. The default is **monk_library/ewhttpnossl**.

### 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 19** for more information about the "load path." (The default is **httpnossl-init.monk**. See **httpnossl-init** on page 58 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 2 on page 13**).

### Startup Function

#### Description

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

#### Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. (The default is **httpnossl-startup**. See **httpnossl-startup** on page 63 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 2 on page 13**). 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 **httpnossl-outgoing**. See **httpnossl-outgoing** on page 61 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 7 on page 18** 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 47 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 **httpnossl-exchange**. See **httpnossl-exchange** on page 57.)

### Additional Information

The function accepts no input and must return a string (see **Figure 5 on page 16** 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 5 on page 16** 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 52 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 **httpnossl-connect**. See **httpnossl-connect** on page 56 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 **httpnossl-verify**. See **httpnossl-verify** on page 64 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 **httpnossl-shutdown**. See **httpnossl-shutdown** on page 62 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 **httpnossl-ack**. See **httpnossl-ack** on page 55 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 **httpnossl-nack** on page 59 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 acknowledgement 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. For more information see **httpnossl-notify** on page 60.

#### Additional Information

When the Control Broker issues a shutdown command to the e*Way, 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 51).

***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 69 for more information.) as well as the **GET** and **POST** calls.(See **Sample Monk Scripts** on page 43 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 URL to which the **GET** or **POST** command will be effected. This URL nominally references a program that will process the **POST**ed data or **GET** request.

### Required Values

A string containing a valid URL address. The user must include the full URL, i.e.,

http://www.stc.com

or

http://www.info.netscape.com/homesearch2.cgi

### Additional Information

If using **GET**, you can provide parameters using the URL encoded query string 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 program requires the data to be encoded in this manner prior to receiving it.

## User Name

### Description

Specifies the username necessary for connecting to the HTTP server.

### Required Values

A string containing any valid username. (See also **Encrypted Password** on page 26 below.)

### Additional information

The username is necessary for authentication purposes.

## Encrypted Password

### Description

Specifies the encrypted password corresponding to the username entered in the User Name field (see above).

### Required Values

A string containing the valid encrypted password associated with the username.

### Additional Information

The username must be defined prior to defining the password.

## Agent

### Description

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

### Required Values

A string. The 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**.

It should not be changed unless there is a specific need to do so.

## 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 URL encoded query string format of name/value pairs, i.e.,

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

## Accept-type

### Description

Specifies the parameters for the Accept-type request header.

### Required Values

A string.

## 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**.

Chapter 4

# Implementation

This chapter includes information pertinent to implementing the HTTP e*Way in a production environment.

## 4.1 Implementation Process: Overview

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

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

- 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 Monk Scripts" on page 43** 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. 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.*

**Note:** *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 from a Command line

**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. Do not specify any file extension— the converter will supply the .ssc extension automatically.

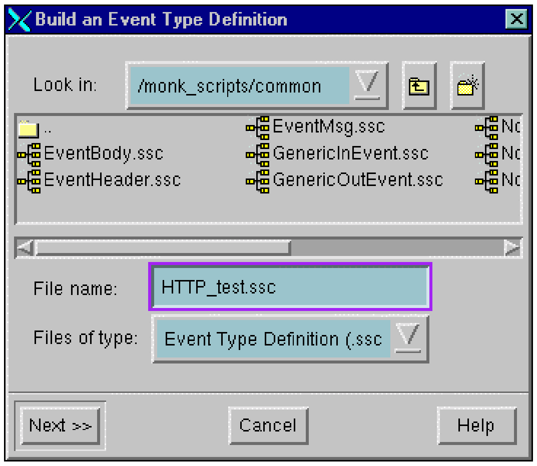
**Note:** *The **output_file** argument must be the last argument listed.*

## 4.2.2 Creating Event Type Definitions from the ETD Editor

**To create an ETD using the HTML Converter from the GUI:**

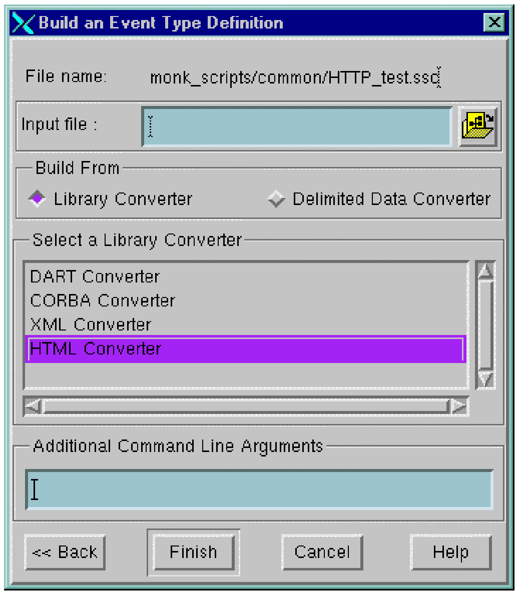
1. Launch the ETD Editor.
2. On the ETD Editor's Toolbar, click **Build**. The **Build an Event Type Definition** dialog box appears.

**Figure 8** 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 9** 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 your input HTML page contains more than one form, the Build tool will create multiple ETD files, one for each form. The name of each file will be the file name you entered in step 3 above with an underscore and number appended to it, 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.

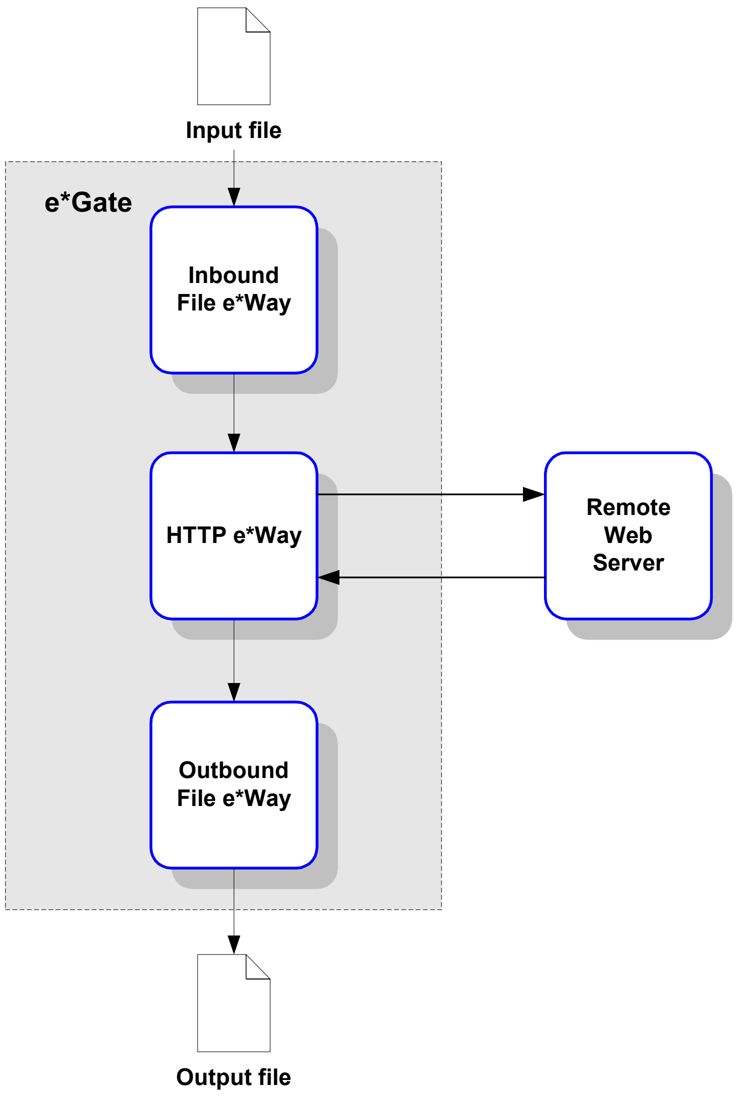
## 4.3 Sample Configurations

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

### 4.3.1 Creating a Schema Using httpnossl-outgoing

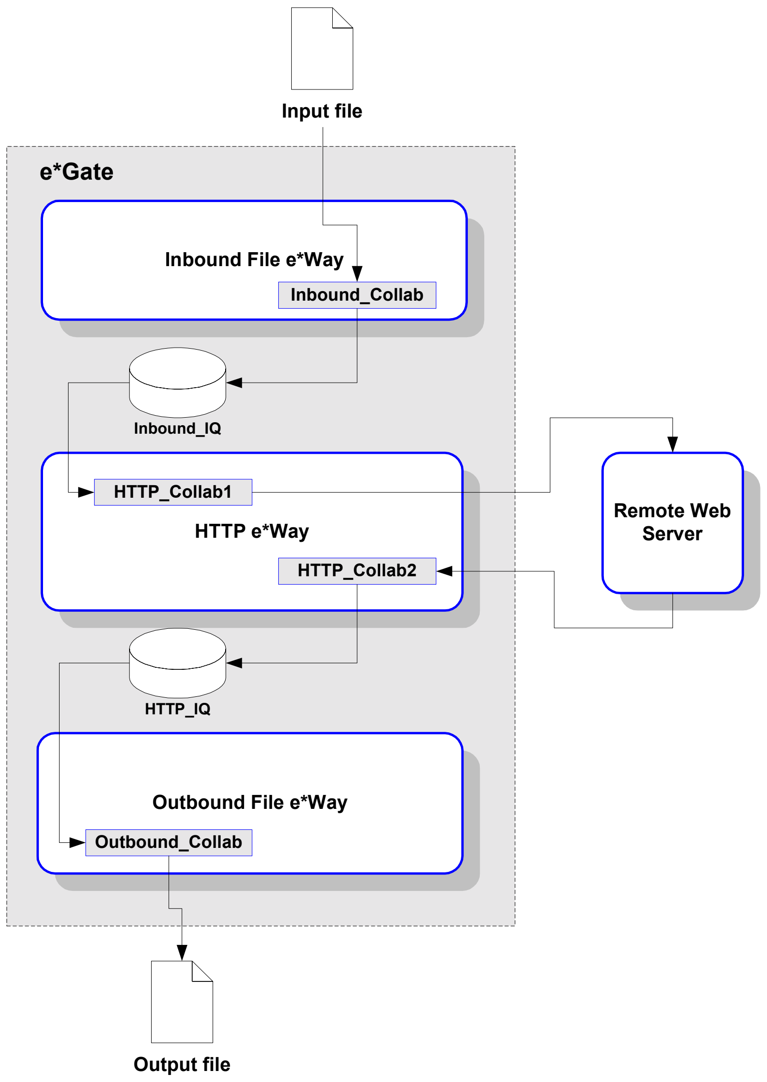
This section demonstrates how to set up a basic schema using the **httpnossl-outgoing** function. In this sample, data is drawn from a text file using the file e*Way and sent to an external system using the HTTP e*Way. The data returned from the external system is received by the HTTP e*Way, then forwarded to another file e*Way and stored in an output file on the local system (see **Figure 10 on page 34**).

**Figure 10** Sample schema: basic architecture



This schema requires a number of components, as illustrated in **Figure 11 on page 35**.

**Figure 11** 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 "http_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 “HTTP_IQ”.

8 Create an e*Way named “Inbound”.

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

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

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

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

| Section | Parameter and setting |
|---|---|
| General Settings | **AllowIncoming**:Yes<br>**AllowOutgoing**:No |
| Poller(inbound) Settings | **Polldirectory**: C:\TEMP (or other “temporary” directory)<br>**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<br>**Source:** <External>. |
| Publications | **Event: In**<br>**Publish to:** Inbound_IQ. |

Now that the “inbound” e*Way is completely configured, you must create an outbound HTTP e*Way.

1 Create a new e*Way component named “http_eway”.

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

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

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

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

5 Save the settings, promote to 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.

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

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

1 Select the **http_eway** component and create a Collaboration named "http_collab1".

2 Assign the following properties to the Collaboration:

| | |
|---|---|
| Collaboration Rules | Passthrough_Data |

| | |
|---|---|
| Subscriptions | **Event:** In<br>**Source:** Inbound_collab |
| Publications | **Event:** In<br>**Publish to:** <External> |

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

4 Assign the following properties to the Collaboration:

| | |
|---|---|
| Collaboration Rules | Passthrough_Data |
| Subscriptions | **Event:** In<br>**Source:** <External> |
| Publications | **Event**: In<br>**Publish to:** HTTP_IQ |

Now create and configure the final e*Way component.

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

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

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

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

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

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

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

7 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the 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<br>**Source**: http_collab2 |
| Publications | **Event**: In<br>**Publish to**: <External> |

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

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

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

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

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

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

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

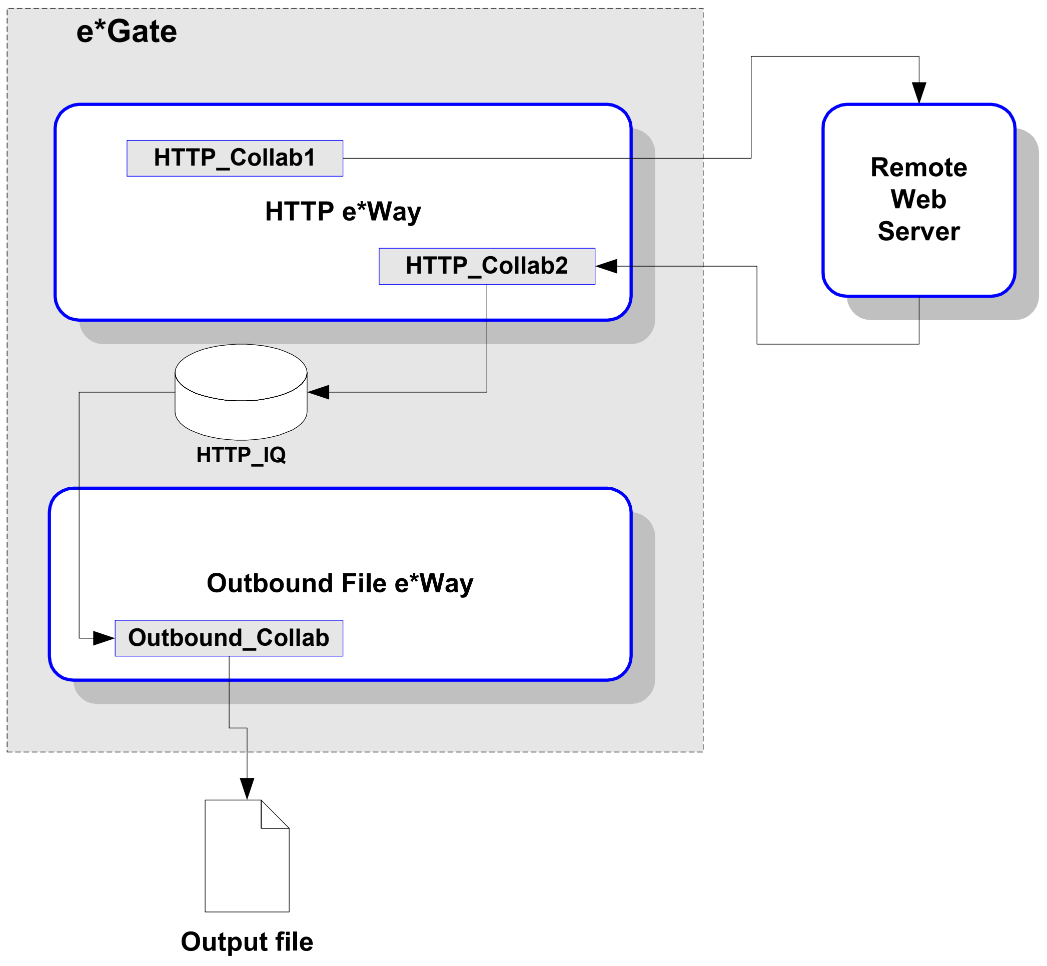
2 Save the file as C:\TEMP\TESTDATA.FIN (if you specified a different input directory, please make the appropriate substitution).

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

## 4.3.2 Creating a Schema Using httpnossl-exchange

This schema, which illustrates the use of the Monk function **httpnossl-exchange**, is simpler than the one illustrated in **"Creating a Schema Using httpnossl-outgoing" on** page 34. Rather than using an inbound e*Way, the data to be sent to the external Web server is hard-coded into the HTTP e*Way's configuration using the e*Way editor. Except for this change, the architecture is the same.

**Figure 12** Sample http-exchange schema



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

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

7 Create an Intelligent Queue, named "HTTP_IQ".

You must create an outbound HTTP e*Way.

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

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

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

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

| Section | Parameter and Settings |
|---|---|
| General Settings | Leave all settings unchanged |
| Communication Setup | **Exchange Data Interval**: 10 (ten)<br>**Zero Wait Between Successful Exchanges**: No |
| Monk Configuration | **Auxiliary Library Directories**: monk_library/ewhttpnossl<br>**Monk Environment Initialization File**: monk_library/ ewhttpnossl:/httpnossl-init.monk<br>**Startup Function**: **h**ttpnossl-startup<br>**Process Outgoing Message Function**: httpnossl-outgoing<br>**Exchange Data With External Function**: httpnossl-exchange<br>**External Connection Establishment Function**: httpnossl-connect<br>**External Connection Verification Function**: httpnossl-verify<br>**External Connection Shutdown Function**: httpnossl-shutdown<br>**Positive Acknowledgment Function**: httpnossl-ack<br>**Negative Acknowledgment Function**: httpnossl-nack<br>The remaining parameters may be left blank for this sample. |
| HTTP Configuration | **Request: GET**<br>**Timeout:** 5000<br>**URL**: enter an appropriate URL to contact.<br>**User Name**: enter an appropriate user name if necessary<br>**Encrypted Password**: enter an appropriate password if necessary<br>**Agent**: e*Gate HTTP e*Way<br>**Content-type**: **Content-Type:**application/x-www-form-urlencoded<br>**Request-content**: Leave this entry blank (because this is a sample using GET; fill in this field when using the POST method).<br>**Accept-type**: **accept:**text/*<br>The remaining parameters may use the default values. |

| Section | Parameter and Settings |
|---|---|
| HTTP Proxy Configuration | Leave blank unless required |

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

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

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

1 From the Enterprise Manager's **File** menu, select **Edit File**.

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

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

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

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

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

2 Assign the following properties to the Collaboration:

| | |
|---|---|
| Collaboration Rules | Passthrough_Data |
| Subscriptions | **Event**: In<br>**Source**: <External> |
| Publications | **Event**: In<br>**Publish to**: HTTP_IQ |

Now create and configure the final e*Way component.

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

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

3 Display the e*Way's properties. Then, use the **Find** button, 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<br>**AllowOutgoing**: Yes |
| Outbound (sender) Settings | **Output directory**: C:\TEMP (or other "temporary" directory)<br>**Output File Name**: httpnossl_out.txt |

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

7 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the 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<br>**Source**: http_collab2 |
| Publications | **Event**: In<br>**Publish to**: <External> |

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

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

## 4.4 Sample Monk Scripts

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

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

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

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

;; Print the results
(display pszData)

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

***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_nossl.dll")

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

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

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

Chapter 5

# HTTP e*Way Functions

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

The HTTP e*Way's 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. If the external collaboration(s) is successful in publishing the Event to the outbound queue, the function will return **#t** (true), otherwise **#f** (false).

### Parameters

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

### Return Values

**Boolean**
Returns **#t** (true) when successful and **#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.

### 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 25**). 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's configuration file. Does not effect 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's configuration file. Execution will be stopped when the e*Way concludes any open transaction. Does not effect any defined schedules, and does not halt the e*Way process itself.

### Parameters

None.

### Return Values

None.

### Throws

None.

## 5.2 HTTP Standard Functions

The current suite of HTTP Monk standard functions are:

### httpnossl-ack

#### Syntax

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

#### Description

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

#### Parameters

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

#### Return Values

**string**

An empty string indicates a successful operation. The e*Way 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 23** for more information.

## httpnossl-connect

### Syntax

```
(httpnossl-connect)
```

### Description

**httpnossl-connect** establishes a connection to the external system.

### Parameters

None.

### Return Values

**string**
"UP" indicates the connection is established. Anything else indicates no connection.

### Throws

None.

### Additional Information

See **"External Connection Establishment Function" on page 22** for more information.

## httpnossl-exchange

### Syntax

```
(httpnossl-exchange)
```

### Description

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

### Parameters

None.

### Return Values

**string**

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

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

“CONNERR” indicates a problem with the connection to the external system. When the connection is re-established this function will be reexecuted with the same input Event.

### Throws

None.

### Additional Information

See **“Exchange Data with External Function” on page 21** for more information.

## httpnossl-init

### Syntax

```
(httpnossl-init)
```

### Description

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

### Parameters

None.

### Return Values

**string**
If a "FAILURE" string is returned, the e*Way will shutdown. 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 19** for more information.

## httpnossl-nack

### Syntax

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

### Description

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

### Parameters

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

### Return Values

**string**

An empty string indicates a successful operation.

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

### Throws

None.

### Additional Information

See **"Negative Acknowledgment Function" on page 24** for more information.

## httpnossl-notify

### Syntax

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

### Description

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

### Parameters

| Name | Type | Description |
|---|---|---|
| command | string | When the e*Way calls this function, it 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 25** for more information.

## httpnossl-outgoing

### Syntax

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

### Description

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

### Parameters

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

### Return Values

**string**

An empty string indicates a successful operation.

"RESEND" causes the Event to be immediately resent.

"CONNERR" indicates a problem with the connection to the external system. When the connection is re-established this function will be reexecuted 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 47 for more information on this function.

### Additional Information

See **"Process Outgoing Message Function" on page 20** for more information.

## httpnossl-shutdown

### Syntax

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

### Description

**httpnossl-shutdown** requests that the external connection shutdown. A return value of "SUCCESS" indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a (**shutdown-request** on page 51) call from within a Monk function to allow the requested shutdown to 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 23**.

## httpnossl-startup

### Syntax

```
(httpnossl-startup)
```

### Description

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

### Parameters

None.

### Return Values

**string**
"FAILURE" causes 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 20** for more information.

## httpnossl-verify

### Syntax

```
(httpnossl-verify)
```

### Description

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

### Parameters

None.

### Return Values

**string**
"UP" or "SUCCESS" if connection established. Anything other value indicates the connection is not established.

### Throws

None.

### Additional Information

See **"External Connection Verification Function" on page 22** 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 79

**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** Unwise Characters

| Unwise 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 | a 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: <br>▪ Accept<br>▪ Accept-Charset<br>▪ Accept-Encoding<br>▪ Accept-Language<br>▪ Authorization<br>▪ Expect<br>▪ From<br>▪ Host<br>▪ If-Match<br>▪ If-Modified-Since<br>▪ If-None-Match<br>▪ If-Range<br>▪ If-Unmodified-Since<br>▪ Max-Forwards<br>▪ Proxy-Authorization<br>▪ Range<br>▪ Referer |
| field_value | string | The field value associated with the header being added. |

### Return Values

**Boolean**

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

## http-clear-content-type-param

### Syntax

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

### Description

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

### Parameters

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

### Return Values

**Boolean**

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

### Throws

None.

## http-clear-headers

### Syntax

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

### Description

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

### Parameters

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

### Return Values

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

### Throws

None.

## http-get

### Syntax

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

### Description

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

### Parameters

| Name | Type | Description |
|---|---|---|
| hCon | opaque handle | The handle provided by **http-acquire-provider**. |
| URL | string | The URL that the **http-get** request is to retrieve when executed. |
| 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.<br>NOTE: Only text types are supported.<br>Must be in the form:<br>Accept:xxxx/xxxx. For example:<br>“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 78 for more information.

### Examples

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

(display pData)
```

### http-get-error-text

#### Syntax

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

#### Description

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

#### Parameters

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

#### Return Values

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

#### Throws

None.

## http-get-last-status

### Syntax

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

### Description

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

### Parameters

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

### Return Values

**integer**
Returns an integer corresponding to specified HTTP server status codes.

**Figure 13** 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** 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****).*

### 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.<br>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 66.

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

### Example One

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

### Example Two

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

## http-release-provider

### Syntax

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

### Description

**http-release-provider** 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. For example: "www.somesite.com" or "www.somesite.com:8080" |
| port | integer | The port number on which the proxy server is listening. |
| proxyUser | string | A valid user name. |
| proxyPassword | string | An encrypted password associated with the above named user. Use the **encrypt-password** function to create this password. See the *Monk Developer's Reference* for more information. |

### Return Values

### Boolean

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

### Throws

None.

## http-url-encode

### Syntax

(http-url-encode *input_data*)

### Description

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

### Parameters

| Name | Type | Description |
|---|---|---|
| input_data | string | The string to be encoded. |

### Return Values

**string**
Returns the encoded string.

### Throws

None.

### Additional Information

In previous releases of the HTTP e*Way this was handled automatically. Currently, this function must be called in order to transform the data string into urlencoded format.

# Index