*SeeBeyond ICAN Suite*

# e-Mail e*Way Intelligent Adapter User's Guide

*Release 5.0.5 for Schema Run-time Environment (SRE)*

*Monk Version*

SEEBEYOND®

# Contents

**Contents**

# Introduction

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

## 1.1 Overview

The e-Mail e*Way enables the e*Gate system to exchange data with an SMTP (outbound) or POP3 (inbound) mail server. The e*Way can therefore upload messages to a mail server and download messages from it.

Functions are provided to log into a server, create e-Mails, add recipients, subject headers, content, and to add attachments. Functions are also provided to read data associated with an incoming message and to save attachments.

Collaborations can therefore be written to intelligently send emails with formatted content and to receive, parse and act upon incoming messages.

### 1.1.1. Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have expert-level knowledge of Windows and UNIX operations and administration; to be thoroughly familiar with SMTP and POP3 mail server protocol and to be thoroughly familiar with Windows-style GUI operations.

### 1.1.2. Components

The following components comprise the e-Mail e*Way:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk function scripts

A complete list of installed files appears in **Table 1 on page 11**.

## 1.2 Supported Operating Systems

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

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

## 1.3 Korean Solaris 8 and 9System Requirements

To use the e-Mail e*Way, you need the following:

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

*Important:* *The e-mail e*Way does not support sending/receiving messages containing multi level nested MIME parts.*

# Installation

This chapter describes how to install the e-Mail e*Way.

## 2.1 Windows Installation

### 2.1.1. Pre-installation

1  Exit all Windows programs before running the setup program, including any anti-virus applications.

2  You must have Administrator privileges to install this e*Way.

### 2.1.2. Installation Procedure

**To install the e-Mail e*Way on a Windows system**

1  Log in as an Administrator on the workstation on which you want to install the e*Way.

2  Insert the e*Way installation CD-ROM into the CD-ROM drive.

3  If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

4  The InstallShield setup application will launch. Follow the on-screen instructions to install the e*Way.

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

5  After the installation is complete, exit the install utility and launch the Schema Designer.

6  In the Components editor, create a new e*Way.

7  Display the new e*Way's properties.

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

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

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

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

   12   The e\*Way Editor will launch. Make any necessary changes, then save the configuration file and promote to run time.

   13   You will return to the e\*Way's property sheet. Click **OK** to close the properties sheet, or continue to configure the e\*Way. Configuration parameters are discussed in **Chapter 3**.

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

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

## 2.2   UNIX Installation

### 2.2.1. Pre-installation

Root privileges are not require to install this e\*Way. Log in under the user name that will 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 e-Mail 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 "e\*Gate Add-on Applications" option. Then, follow any additional on-screen directions.

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

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

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

8  Display the new e\*Way's properties.

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

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

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

12  From the **Select an e\*Way Template** list, select **stcewemail** and click **OK**.

13  The e\*Way Editor will launch. Make any necessary changes, then save the configuration file and promote to run time.

14  You will return to the e\*Way's property sheet. Click **OK** to close the properties sheet, or continue to configure the e\*Way. Configuration parameters are discussed in **Chapter 3**.

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

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

## 2.3 Files/Directories Created by the Installation

The e-Mail 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_monkemail.dll |
| configs\stcewgenericmonk\ | stcewemail.def |
| monk_library\ | ewemail.gui |
| monk_library\ewemail\ | email-verify.monk<br>email-startup.monk<br>email-shutdown.monk<br>email-outgoing.monk<br>email-notify.monk<br>email-nack.monk<br>email-init.monk<br>email-extconnect.monk<br>email-exchange.monk<br>email-ack.monk |

# Configuration

This chapter describes how to configure the e-Mail e*Way.

## 3.1  e*Way Configuration Parameters

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

**To set or change e*Way configuration parameters**

1  In the Schema Designer's Components tab, select the e*Way you want to configure and display its properties.

2  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.

3  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.

*Note:  When creating a new e*Way, you must also select **stcewemail**.*

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *Working with e*Ways* chapter of the *e*Way Integrator User's Guide*.

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

- General Settings
- Communication Setup
- Monk Configuration
- e-Mail Configuration

### 3.1.1.  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 13** for more information.)

# Max Resends Per Message

## Description

Specifies the number of times the e*Way will attempt to resend a message (Event) to the external system after receiving an error.

## Required Values

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

# Max Failed Messages

## Description

Specifies the maximum number of failed messages (Events) 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

Specifies whether error messages received from the external system beginning with the string "DATAERR" will be queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 27** for more information.

## Required Values

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

See **"Schedule-driven data exchange functions" on page 22** for information about how the e*Way uses this function.

## 3.1.2. Communication Setup

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

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

## Exchange Data Interval

### Description

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

### Required Values

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

### Additional Information

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

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

See **"Down Timeout" on page 16** and **"Stop Exchange Data Schedule" on page 16** for more information about the data-exchange schedule.

## Zero Wait Between Successful Exchanges

### Description

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

### Required Values

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

See **"Exchange Data with External Function" on page 27** 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 repeatedly (every n seconds, minutes, hours, days, or weeks). Since months vary in length, "Months" is not provided as a unit under "Repeatedly."

  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).

**Figure 1**  Add Schedule Dialog Box
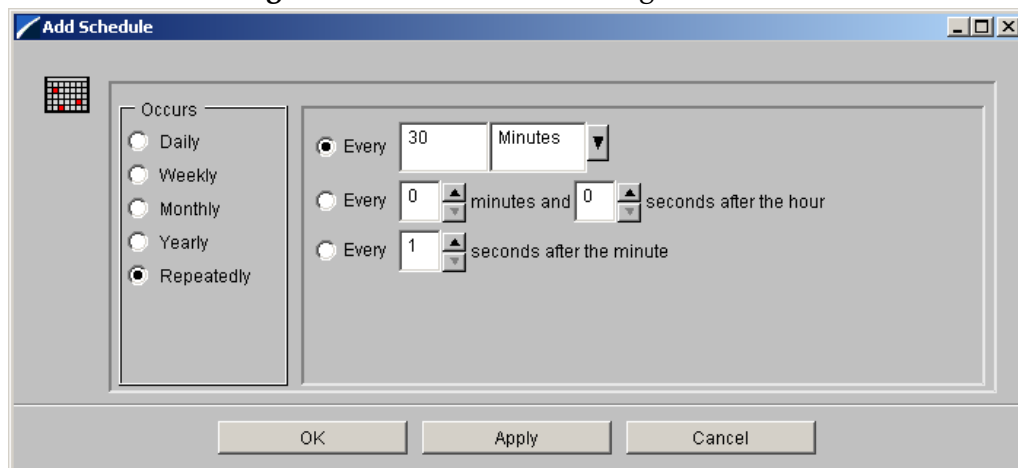


**Also required:** If you set a schedule using this parameter, you must also define all three of the following:

- Exchange Data With External Function

- Positive Acknowledgment Function

- Negative Acknowledgment Function

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

**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 27**, **"Exchange Data Interval" on page 14**, and **"Stop Exchange Data Schedule" on page 16** 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 repeatedly (every n seconds, minutes, hours, days, or weeks). Since months vary in length, "Months" is not provided as a unit under "Repeatedly."(See **Figure 1 on page 15**)

  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 28** 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 28** 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 (Event) to the external system, after receiving an error message from the external system.

**Required Values**

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

### 3.1.3. Monk Configuration

The parameters in this section assist in setting up 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 2 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 2** e*Way internal architecture
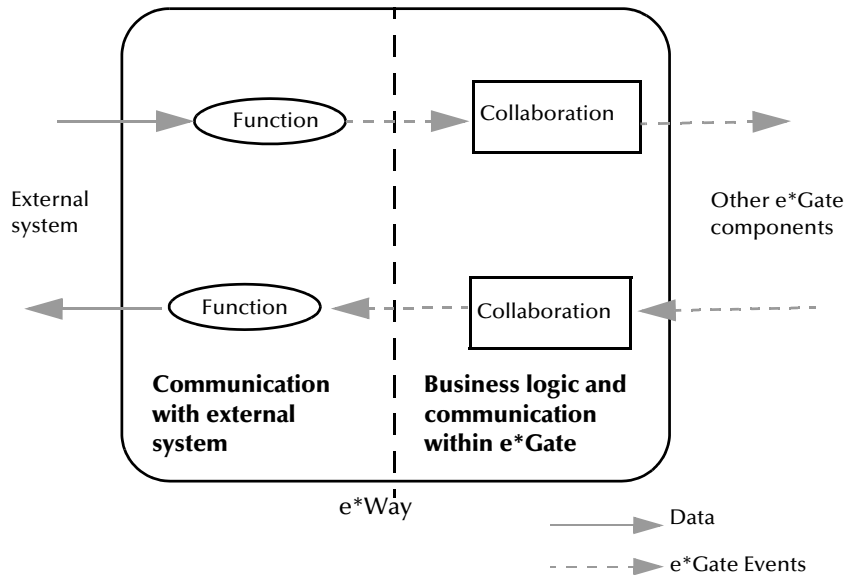


The "communications half" of the e*Way uses Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events" and send those Events to Collaborations, and manage the connection between the e*Way and the external system. The **Monk Configuration** 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 Collaboration Rules Editor or a text editor (such as **write**, **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 26 <br>(also see **Monk Environment Initialization File** on page 25) |
| Connection | **External Connection Establishment Function** on page 28 <br>**External Connection Verification Function** on page 28 <br>**External Connection Shutdown Function** on page 29 |
| Schedule-driven data exchange | **Exchange Data with External Function** on page 27 <br>**Positive Acknowledgment Function** on page 29 <br>**Negative Acknowledgment Function** on page 30 |
| Shutdown | **Shutdown Command Notification Function** on page 30 |
| Event-driven data exchange | **Process Outgoing Message Function** on page 26 |

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

### Initialization Functions

Figure 3 illustrates how the e*Way executes its initialization functions.

**Figure 3**   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 │
                  └─────────────────┘
```
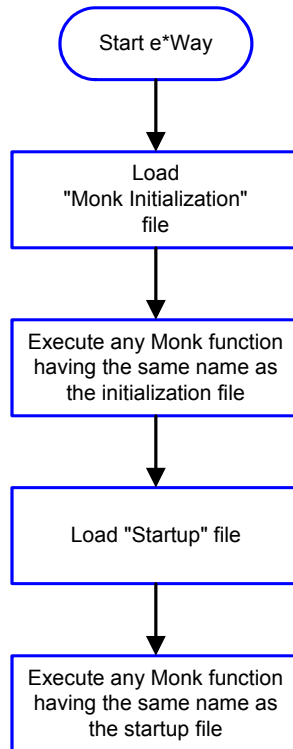
**Connection Functions**

Figure 4 illustrates how the e*Way executes the connection establishment and verification functions.

**Figure 4**  Connection establishment and verification functions
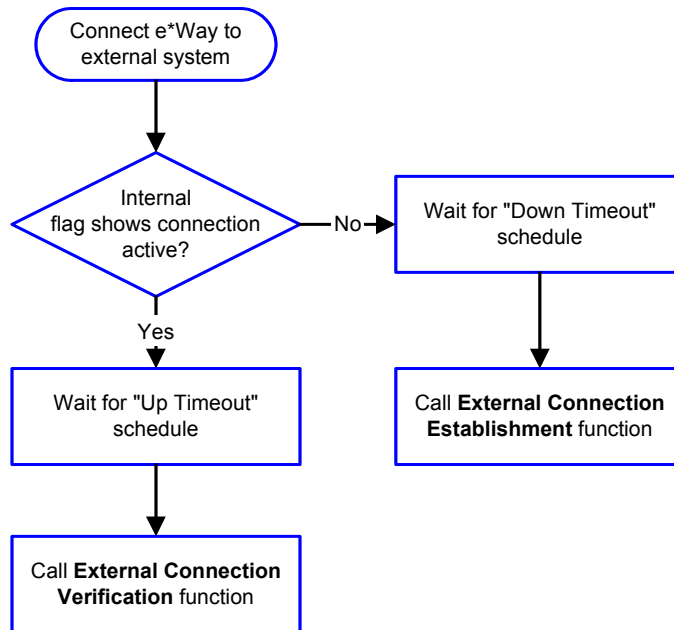


*Note:*  *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 6 on page 22** *and* **Figure 8 on page 24** *for examples of how different functions use this flag.*

*User functions can manually set this flag using Monk functions. See* **send-external-up** *on page 59 and* **send-external-down** *on page 60 for more information.*

Figure 5 illustrates how the e*Way executes its "connection shutdown" function.

**Figure 5** Connection shutdown function
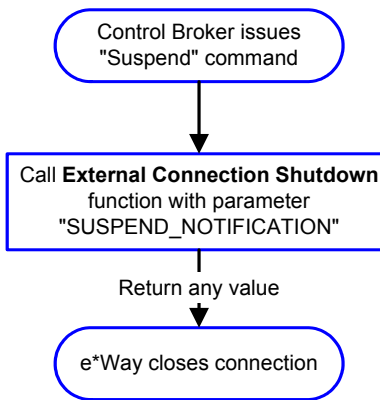


Schedule-driven Data Exchange Functions

Figure 6 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgment Function** and **Negative Acknowledgment Function** are also called during this process.

"Start" can occur in any of the following ways:

- The "Start Data Exchange" time occurs

- Periodically during data-exchange schedule (after "Start Data Exchange" time, but before "Stop Data Exchange" time), as set by the Exchange Data Interval

- The **start-schedule** Monk function is called

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

**Figure 6** Schedule-driven data exchange functions
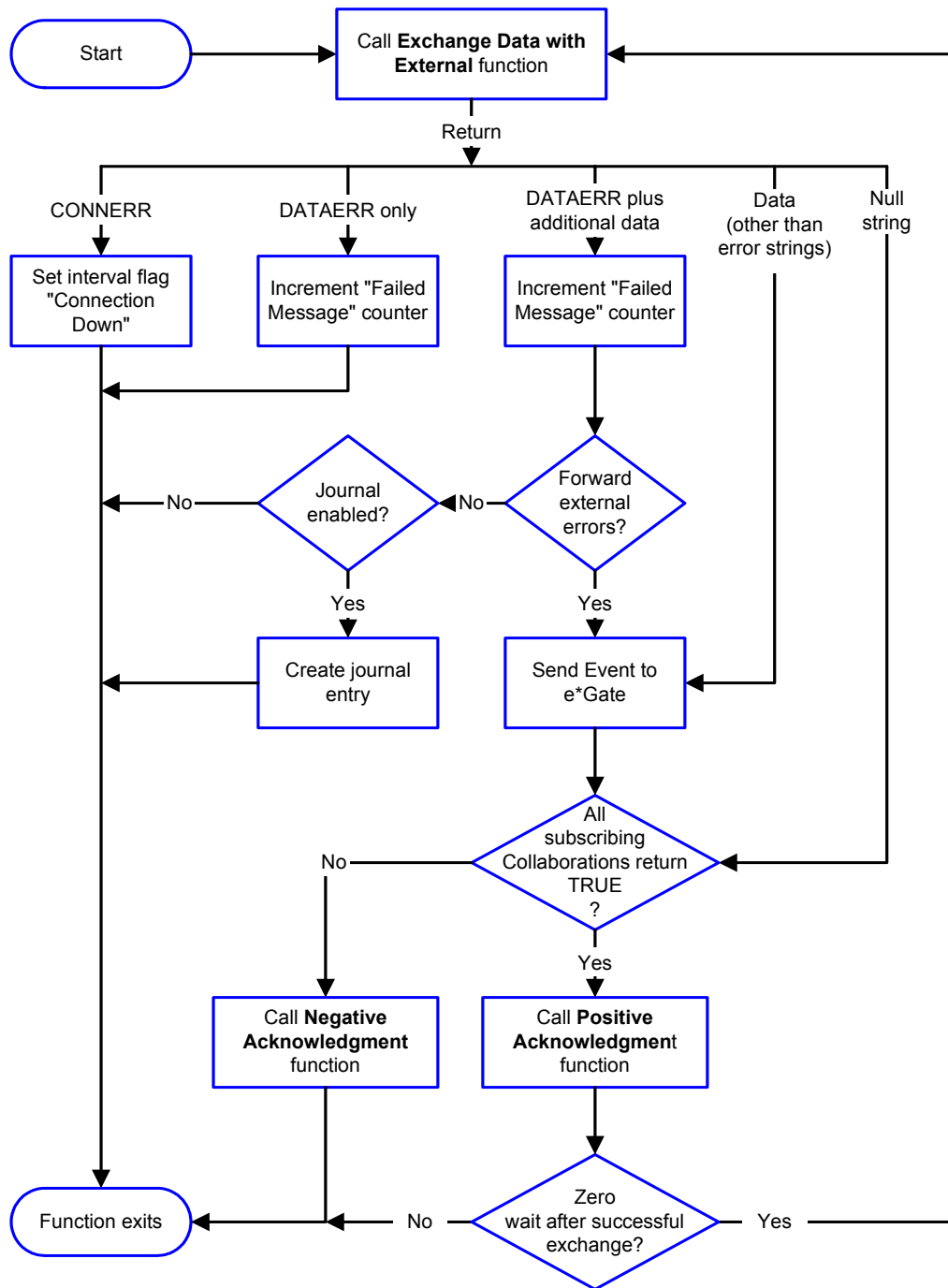
### Shutdown Functions

Figure 7 illustrates how the e*Way implements the shutdown request function.

**Figure 7**   Shutdown functions
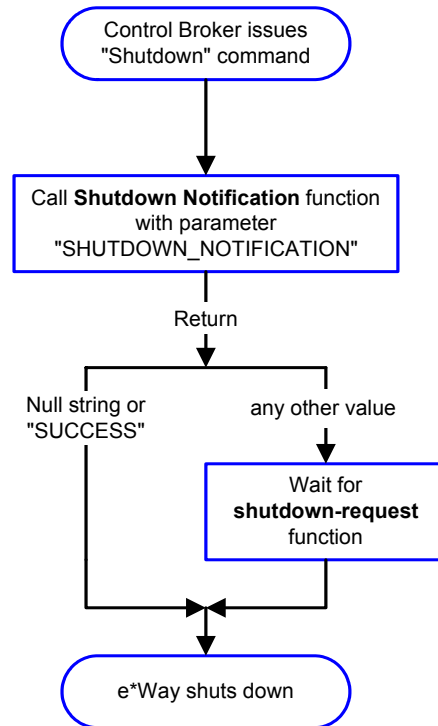


### Event-driven Data Exchange Functions

Figure 8 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 8**  Event-driven data-exchange functions
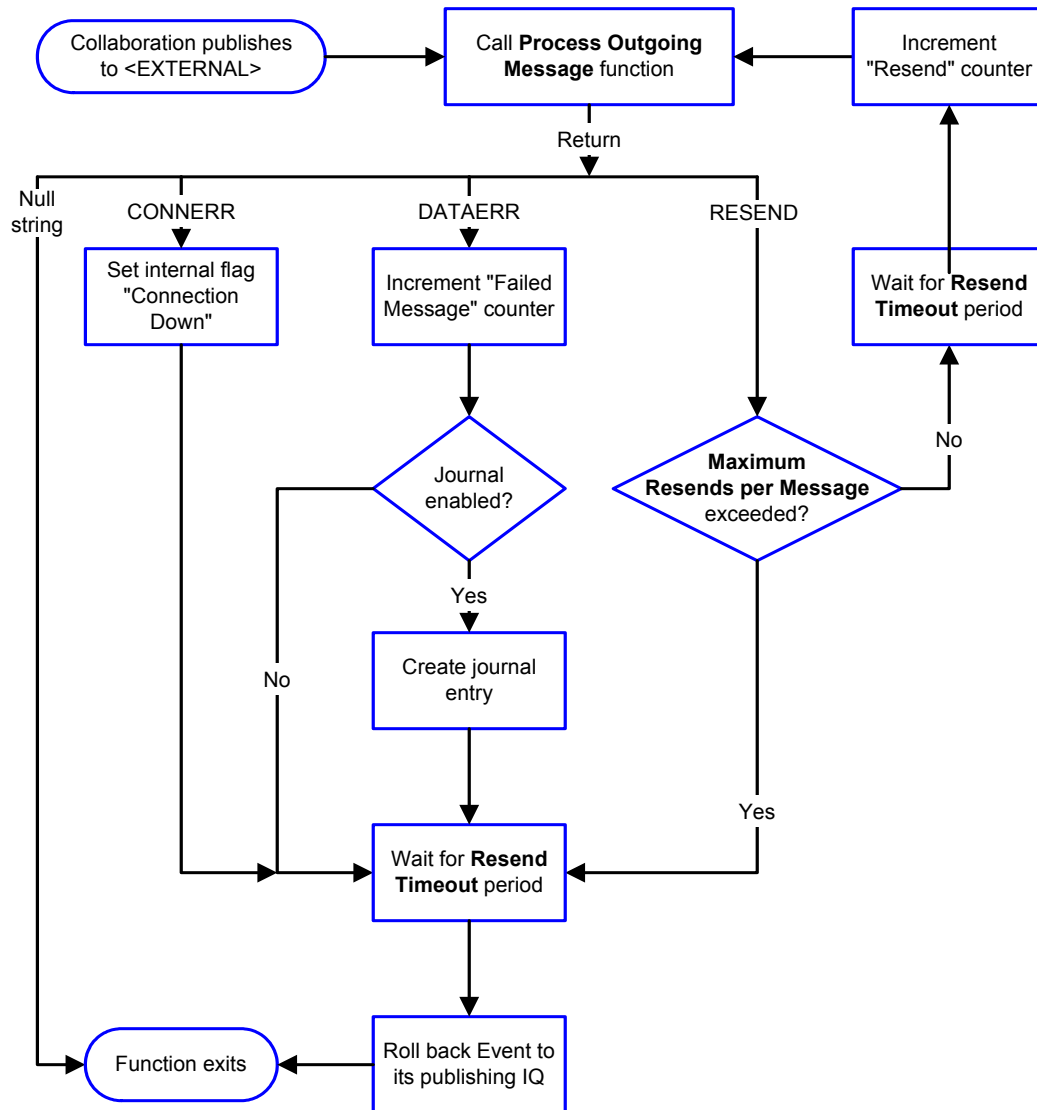


## How to Specify Function Names or File Names

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

- .monk (monk function file.)
- .tsc (collaboration script.)
- .dsc (collaboration script.)

## Additional Path

### Description

Specifies a path to be appended 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 after the default load paths.

### 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 **Find File** 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. This parameter is optional and may be left blank.

### 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
```

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

## Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which will be loaded after the auxiliary library directories are loaded. Use this feature to initialize the e*Way's Monk environment (for example, to define Monk variables that are used by the e*Way's function scripts).

**Required Values**

A filename within the "load path", or filename plus path information (relative or absolute). If additional path information is specified, that path will be appended to the "load path." See **"Additional Path" on page 25** for more information about the "load path." The default is **email-init**. See **email-init** on page 68 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 3 on page 19**).

## Startup Function

**Description**

Specifies a Monk function that the e*Way will load and invoke upon startup or whenever the e*Way's configuration is reloaded. This function should be used to initialize the external system before data 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 **email-startup**. See **email-startup** on page 73 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 Library 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 3 on page 19**). 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. The default is **email-outgoing**. See **email-outgoing** on page 71 for more information. *You may not leave this field blank.*

**Additional Information**

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

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Schema Designer). The function returns one of the following (see **Figure 8 on page 24** 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 62 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 **email-exchange**. See **email-exchange** on page 66 for more information. This parameter is optional and may be left blank.

**Additional Information**

The function accepts no input and must return a string (see **Figure 6 on page 22** 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 6 on page 22** 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 Exchange Data** 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 57 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. The default is **email-extconnect**. See **email-extconnect** on page 67 for more information. *This field cannot be left blank.*

### 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 **email-verify**. See **email-verify** on page 74 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 **email-shutdown**. See **email-shutdown** on page 72 for more information. This parameter is optional.

**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.

## 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 **email-ack**. See **email-ack** on page 65 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: Indicates that 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).

## 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. The default is **email-nack**. See **email-nack** on page 69 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: Indicates that 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).

## 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 required.

**Required Values**

The name of a Monk function. The associated function is **email-notify**. See **email-notify** on page 70 for more information.

**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 63 for more information.

*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.*

## 3.1.4. e-Mail Configuration

The parameters in this section help to set up the required information for the e-Mail e*Way to access the e-Mail server.

## UserName

**Description**

Specifies the name of the owner of the e-Mail account.

**Required Values**

A string containing any valid username. This parameter is *mandatory*.

## Encrypted Password

**Description**

Specifies the password associated with the specified **Username**.

**Required Values**

A string containing any valid password.

**Additional Information**

The **Username** must be defined prior to the **Password**.

## eMailAddress

**Description**

Specifies the e-Mail address of the account.

**Required Values**

A string containing the valid e-Mail address of the account to be accessed. This parameter is *mandatory*.

## ReplyAddress

**Description**

Specifies the address to which reply messages should be sent.

**Required Values**

A string containing the valid e-Mail address to which replies should be sent. This parameter is *mandatory*.

## OutboundServer

**Description**

Specifies the outbound SMTP server.

**Required Values**

A valid SMTP server address. This parameter is *mandatory*.

## OutboundPort

**Description**

Specifies the port on which the SMTP server is listening.

**Required Values**

An integer. The range is 1 to 864000. The configured default is 25. This parameter is *mandatory*.

## InboundServer

**Description**

Specifies the inbound POP3 server.

**Required Values**

A valid POP3 server address. This parameter is *mandatory*.

## InboundPort

**Description**

Specifies the port on which the POP3 server is listening.

**Required Values**

An integer. The range is 1 to 864000. The configured default is 110. This parameter is *mandatory*.

# Implementation

This chapter contains information pertinent to implementing the e-mail e*Way in a production environment. Also included is a sample schema. This chapter assumes that the e-Mail e*Way has been successfully installed, and that the executable and configuration files have been appropriately assigned.

## 4.1 Implementation Overview

During installation, the Participating Host (host) and Control Broker are automatically created and configured. The default name for each is the name of the host on which you installed the e*Gate Schema Designer GUI. For example, *localhost* and *localhost_cb*.

To complete the implementation of the e-Mail e*Way, do the following:

- Verify that **stc_monkemail.dll** has been installed. See **"Files/Directories Created by the Installation" on page 11** for details. Ensure the Control Broker is activated.

- In the e*Gate Schema Designer define and configure the following:

  - The e*Way components.

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

  - Collaboration Rules to process Events.

  - Collaborations, within the e*Way components, to apply the required Collaboration Rules.

  - Intelligent Queues (IQs) to which data will be published prior to being sent to the external system.

**To create e*Way components**

1 Select the Navigator's Components tab.

2 Open the host on which you want to create the e*Way.

3 Select the Control Broker that will manage the new e*Way.

4 On the Palette, click  .

5 Enter the name of the new e*Way, then click **OK**.

6   Select the new e*Way, then click ![icon] to edit its properties.

7   When the e*Way Properties window opens, click on the **Find** button beneath the *Executable File* field, and select stcewgenericmonk.exe.

8   Under the *Configuration file* field, click the **New** button. When the Settings page opens, set the configuration parameters for this configuration file.

9   Exit from Settings, and save the file.

**To create Event Type Definitions**

There are two ways to create an ETD. You can use the ETD Editor to open an existing .ssc file and modify it as needed, or you can create a new file. For the purpose of this example, the following procedure shows how to create a new ETD

*Note:* *For some e*Gate e*Ways, the ETD Editor Build Tool automatically creates the definitions. However, the e-Mail e*Way does not use this tool.*

1   Click **Options** on the menu bar. Select Default Editor from the drop down menu. Two choices are available for Default Editor, Java or Monk. In this case select Monk.

2   Highlight the **Event Types** folder on the **Components** tab of the e*Gate Navigator.

3   On the Palette, click ![icon] to create a new Event Type.

4   Enter the name of the Event, then click **OK.**

5   Select the new Event Type, then click ![icon] to edit its properties.

6   When the Properties window opens, click the **New** button.

7   When ETD Editor opens, enter the a name for the .ssc file.

An Event Type Definition is a graphical representation of the layout of data in an event. Below is an example of the ETD for an associated Event.

**Figure 9**   Sample ETD File
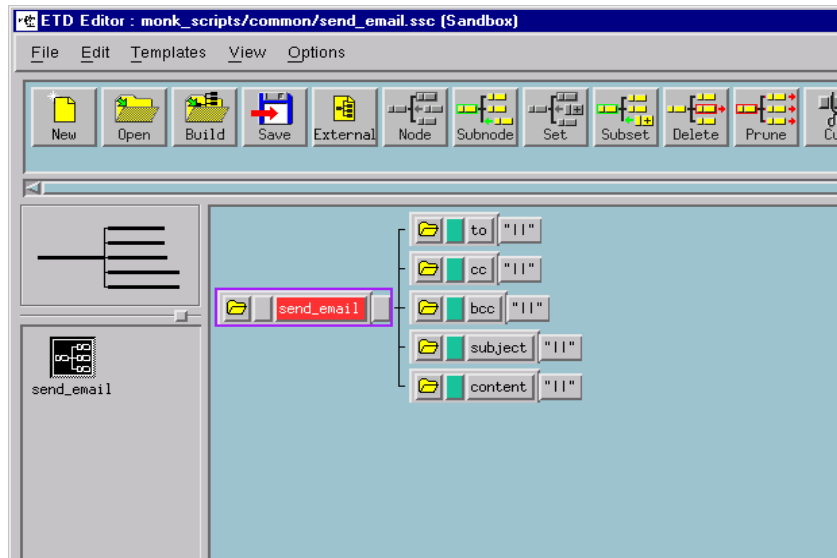


The example above consists of the root node and five subnodes. Refer to the on-line help for details about using the GUI to create the structure.

**8**   Save the file and exit from the ETD Editor.

**9**   When you return to the Event Type Properties tab, click **OK**.

**To create the Collaborations Rules**

The next step is to create the Collaboration Rules that will extract selected information from the source Event Type defined above. The Collaboration Rule will then process the extracted information according to its associated Collaboration Service.

**1**   Select the Navigator's **Components** tab in the e*Gate Schema Designer.

**2**   In the Navigator, select the **Collaboration Rules** folder.

**3**   On the Palette, click [icon] .

**4**   Enter the name of the new Collaboration Rule, then click **OK.**

**5**   Select the new Collaboration Rule, then click [icon] to edit its properties.

**6**   In the **Service** box, select the name of the Collaboration Service that the Collaboration Rules will use, from among the following:

**A**   **Pass Through**: Copies an Event without performing any processing. Select this Collaboration Service when you need to pass an Event unchanged from one component to another.

**B**   **Route Table**: The route table file (*.rtb) provides for backwards compatibility for e*Gate's previous versions, 3.x and lower.

    C  **Monk ID**: Verifies an Event Type against a set of rules. Use this Collaboration Service on inbound e*Ways to verify that inbound Events match known Event Type Definitions.

    D  **Monk**, **Java**, and **C**: Provide support for applications that manipulate Events or Event data.

7  In the **Initialization string** box, enter any required initialization string that the Collaboration Service may require. This field can be left blank.

8  Under **Collaboration Rules**, click **Clear** to clear the file name selection. Click **Find** to select the file containing the Collaboration Rules. Click **Edit** to modify the file. If there is no file selected, click **New** to create a new one. Click **Test** to test the syntax and functionality of the Collaboration Rules file (available for Rules using the Monk or Monk ID service only).

9  Under **Initialization file**, click **Find** to select the file containing the initialization parameters for the Collaboration Service (for example, a Monk initialization file). Click **Clear** to clear the file name selection.

**To create Collaborations**

Collaborations are e*Way components that receive and process Event Types, then forward the output to other e*Gate components. Collaborations consist of the Subscriber, which "listens" for Events of a known type (sometimes from a given source), and the Publisher, which distributes the transformed Event to a specified recipient. The same Collaboration cannot be assigned to more than one e*Gate component.

1  In the e*Gate Schema Designer, select the **Components** tab.

2  Open the host on which you want to create the Collaboration.

3  Select a Control Broker.

4  Select the newly created e-Mail e*Way to assign the Collaboration.

5  On the Palette, click  .

6  Enter the name of the new Collaboration, then click **OK.**

7  Select the new Collaboration, then click  to edit its properties.

8  From the **Collaboration Rules** list, select the Collaboration Rules file that you created previously.

9  In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

    A  From the **Event Type** list, select the Event Type that you previously defined.

    B  Select the subscription source from the **Source** list. In this case, it should be <EXTERNAL>.

10  In the **Publications** area, click **Add** to define the output Event Types that this Collaboration will publish.

A   From the **Event Types** list, select the Event Type that you previously defined.

B   Select the publication destination from the **Destination** list. In this case, it should be <EXTERNAL>. You can publish to an IQ but you cannot subscribe to one.

11   Click **Advanced** to set additional properties for the Collaboration.

**To create Intelligent Queues**

The final step is to create and associate an Intelligent Queue (IQ) for the e-Mail e*Way. IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

1   Select the Navigator's **Components** tab.

2   Open the host on which you want to create the IQ.

3   Open a Control Broker.

4   Select an IQ Manager.

5   On the Palette, click  .

6   Enter the name of the new IQ, then click **OK.**

7   Select the new IQ, then click  to edit its properties.

8   On the General Tab, specify the Service and the Event Type Get Interval.

The STC_Standard IQ Service provides sufficient functionality for most applications. If specialized services are required, custom IQ Service DLLs may be created.

The default *Event Type Get Interval* of 100 milliseconds is satisfactory for the purposes of this initial implementation.

9   On the **Advanced** tab, make sure that *Simple publish/subscribe* is checked under the **IQ behavior** section.

## 4.2   Sample Schema

The previous sections provided the basic procedures for implementing the e-Mail e*Way. This section describes how to use the e-Mail e*Way within a sample schema. The sample will send and receive e-Mail from any platform file. It is assumed that the e-Mail e*Way has been installed properly, and that all of the necessary files and scripts are in the default location.

The sample schema follows these steps:

- ▪ **"Create the Schema" on page 39**

- ▪ **"Add and Configure the e*Ways" on page 40**

- ▪ **"Create and Configure the Event Types" on page 45**

- ▪ **"Create the Collaboration Rules" on page 47**

- ▪ **"Add and Configure the Collaborations" on page 49**

- ▪ **"Run the Schema" on page 51**

This implementation will consist of four e*Ways, two Event Types, two Collaboration Rules, two Intelligent Queues and four Collaborations, as follows:

**e*Ways**

- ▪ **Inbound** - This e*Way will receive input from an external source, apply Collaboration Rules, and publish the information to an Intelligent Queue.

- ▪ **Outbound** - This e*Way processes the external Event and copies the information to an output file.

- ▪ **Send_Mail** - This e*Way applies Collaboration Rules to an inbound Event before it is e-mailed to external recipients.

- ▪ **Receive_Mail** - This e*Way receives an Event from an external e-Mail account, and copies the e-mail to an Intelligent Queue.

**Event Types**

- ▪ **Sort_Send** - This Event Type describes an Event that is input to the Inbound Collaboration.

- ▪ **Sort_Receive** - This Event Type defines an Event from an external e-Mail account.

**Collaboration Rules**

- ▪ **crPass_Mail** - This Collaboration Rule is associated with the **Sort_Send** Event Type, and is used for forwarding e-mail to external recipients.

- ▪ **crReceive_Mail** - The Collaboration Rule is associated with the **Sort_Receive** Event Type for input, and is used for processing external e-mail.

**Intelligent Queues**

- ▪ **iq_Send** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the **Send_Mail** e*Way.

- ▪ **iq_Receive** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the **Outbound** e*Way.

**Collaborations**

- ▪ **col_inbound** - This Collaboration will be a member of the **Inbound** e*Way, applying the **crPass_Mail** Rules, will contain the **Sort_Send** Event Type for input and publishes to the **iq_send** IQ.

- ▪ **col_Outbound** - This Collaboration will be a member of the **Outbound** e*Way, applying the **receive_mail** Rules, and will contain the **sort_receive** Event Type for output.

▪ **col_Receive_mail** - This Collaboration will be a member of the **Receive_Mail** e*Way, applying the **receive_mail** Rules, and will contain the **sort_receive** Event Type for publishing to the **iq_Receive** IQ.

▪ **col_Send_mail** - This Collaboration will be a member of the **Send_Mail** e*Way, applying the **crPass_Mail** Rules, and will contain the **Sort_Send** Event Type for forwarding data to external e-mail recipients.

**Figure 10**   Sample e-Mail Schema
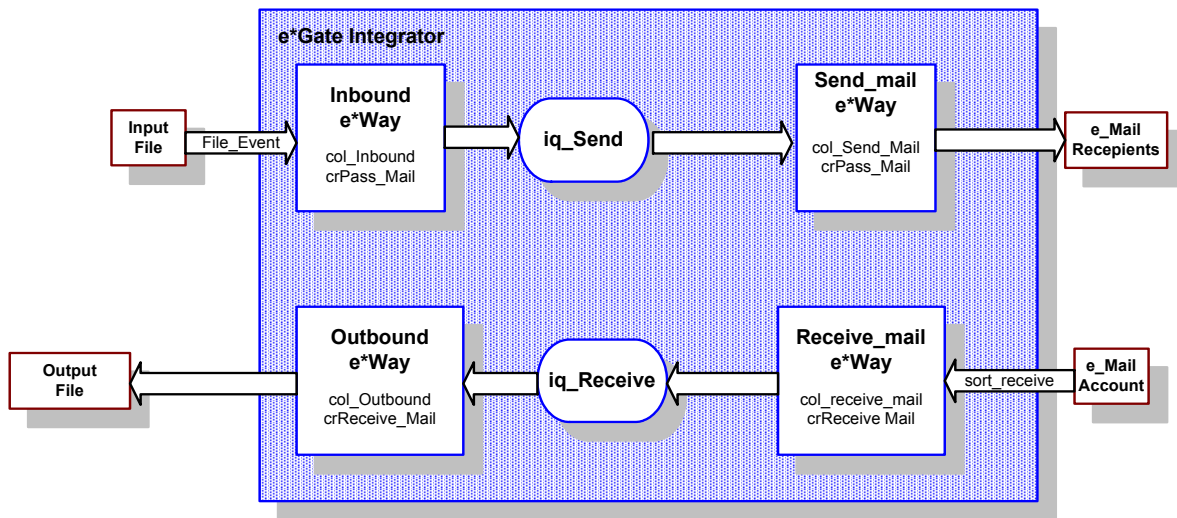


This sample will send, or receive and forward, an e-mail message. In the first example, the external source is a simple e-mail, received by the **Inbound** e*Way, published to the **iq_Send** IQ, then forwarded to external e-mail recipients through the **Send_mail** e*Way.

The second example uses an external e-mail as the source which is received by the **Receive_Mail** e*Way, published to the **iq_Receive** IQ, then stored in an external file through the **Outbound** e*Way. The sample will also verify that the e-Mail e*Way has been properly installed and configured.

## 4.2.1. Create the Schema

The first task in deploying the sample implementation is to create a new schema. While it is possible to use the default schema for the sample implementation, it is recommended that you create a separate schema for testing purposes. After you install the e-Mail e*Way Intelligent Adapter, do the following:

1   Start the e*Gate Schema Designer GUI.

2   When the Schema Designer prompts you to login, select the Registry Host, User Name, and Password to be used to log in and click **Log In**.

3   From the list of schemas, click **New** to create a new schema.

4   For this sample implementation, enter the name **eMail_Test** and click **Open**.

The Schema Designer will start and display the newly created schema.

You are now ready to begin creating the necessary components for this sample schema.

## 4.2.2. Add and Configure the e*Ways

The sample schema uses four e*Ways: **Inbound**, **Outbound**, **Send_Mail**, and **Receive_Mail**. See **"Sample Schema" on page 37** for more information.

The following sections provide instructions for adding and configuring each e*Way.

**To add and configure the Inbound e*Way**

1 In the Components pane of the Schema Designer, select the Control Broker and

 click ![icon] to add a new e*Way.

2 Enter **Inbound** for the component name and click **OK**.

3 Select the newly created e*Way and click ![icon] to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 3.

**Table 2**   Inbound e*Way Parameters

| Parameter | Value |
|---|---|
| **General Settings (if not otherwise stated use the default setting)** | |
| AllowIncoming | Yes |
| AllowOutgoing | No |
| **Outbound (send) Settings** | Default |
| **Poller (Inbound) Settings** | |
| PollDirectory | /egate/data (input file folder) |
| InputFileMask | *.dat (input file extension) |

7 Select **Save** from the **File** menu. Enter **Inbound** as the file name and click **Save**.

8 Select **Promote to Run Time** from the **File** menu. Click **OK** to promote to run time.

9 A message will notify you that the file has been promoted to run time. Click **OK** to close the e*Way configuration file editor.

10 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

11 Click **OK** to save the e*Way properties.

**To add and configure the Outbound e*Way**

1   In the components pane of the Schema Designer, select the Control Broker and click

     [icon] to add a new e*Way.

2   Enter **Outbound** for the component name and click **OK**.

3   Select the newly created e*Way and click [icon] to display the e*Way's properties.

4   Use the **Find** button to select **stcewfile.exe** as the executable file.

5   Click **New** to create a new configuration file.

6   Enter the parameters for the e*Way as shown in Table 3.

**Table 3**   Outbound e*Way Parameters

| Parameter | Value |
|---|---|
| **General Settings** | |
| AllowIncoming | No |
| AllowOutgoing | Yes |
| **Outbound (send) Settings** | |
| OutputDirectory | /egate/data (output file folder) |
| OutputFileName | output%d.dat (set file name) |

*Note:*   *Parameters not specified in the above table are to retain the default values.*

7   Select **Save** from the **File** menu. Enter **Outbound** as the file name and click **Save**.

8   Select **Promote to Run Time** from the **File** menu. Click **OK** to promote to run time.

9   A message will notify you that the file has been promoted to run time. Click **OK** to close the e*Way configuration file editor.

10  In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

11  Click **OK** to save the e*Way properties.

**To add and configure the Send_Mail e*Way**

This e*Way uses several Monk scripts that are included in the e*Gate installation. Samples of these scripts are shown in the section **"Sample Monk Scripts" on page 52**.

However, should your implementation require additional or different functionality than those in the Monk scripts provided, you can construct your own scripts.

You can create a Monk script with any ASCII text editor. After you finish composing this script, save the file into the **egate/client/monk_scripts/common** folder.

*Note:* *After you compose and save the required Monk script, you must select* **File,**
**Commit to Sandbox** *from the e\*Gate Schema Designer so that the script will be
available for your use.*

Upon completing the Monk script and committing it to the Sandbox, you can create and
configure the e\*Way as follows:

1 In the components pane of the Schema Designer, select the Control Broker and click

 to add a new e\*Way.

2 Enter **Send_Mail** for the component name and click **OK**.

3 Select the newly created e\*Way and click  to display the e\*Way's properties.

4 Use the **Find** button to select **stcewgenericmonk.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Select **stcewemail** in the e\*Way Template Selection window and click **OK**.

7 Enter the parameters for the e\*Way as shown in Table 4.

**Table 4** Send_Mail e\*Way Parameters

| Parameter | Value |
|---|---|
| **General Settings** | Default |
| **Communication Setup** | |
| Exchange Data Interval | 20 |
| Zero Wait between successful Exchanges | Default |
| Start Exchange Data Schedule | blank |
| Stop Exchange Data Schedule | blank |
| Down Timeout | Default |
| Up Timeout | Default |
| Resend timeout | Default |
| **Monk Configuration** | |
| Additional Path | Blank |
| Auxiliary Library Directories | monk_library/ewemail (default) |
| Monk Environment Initialization File | email-init.monk (default) |
| Startup Function | email-startup (default) |
| Process Outgoing Message Function | email-outgoing (default) |
| Exchange Data With External Function | email-exchange (default) |
| External Connection Establishment Function | email-extconnect (default) |
| External Connection Verification Function | email-verify (default) |
| External Connection Shutdown Function | email-shutdown (default) |

**Table 4**  Send_Mail e*Way Parameters

| Parameter | Value |
|---|---|
| Positive Acknowledgment Function | email-ack (default) |
| Negative Acknowledgment Function | email-nack (default) |
| Shutdown command notification function | blank |
| Email Setup (must fill up the information for the following fields): | User Name: (Account to send from)<br>Encrypted Password<br>Email Address: (Account to send from)<br>Reply Address: (Account to send from)<br>Outbound Server: (Name of outbound email server)<br>Outbound Port: (Non-conflicting port number)<br>Inbound Server: Blank<br>Inbound Port: Blank |

8 Select **Save** from the **File** menu. Enter **Send_Mail** as the file name and click **Save**.

9 Select **Promote to Run Time** from the **File** menu. Click **OK** to promote to run time.

10 A message will notify you that the file has been promoted to run time. Click **OK** to close the e*Way configuration file editor.

11 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

12 Click **OK** to save the e*Way properties.

**To add and configure the Receive_Mail e*Way**

This e*Way uses several Monk scripts that are included in the e*Gate installation. Samples of these scripts are shown in the section **"Sample Monk Scripts" on page 52**.

However, should your implementation require additional or different functionality than those in the Monk scripts provided, you can construct your own scripts.

You can create a Monk script with any ASCII text editor. After you finish composing this script, save the file into the **egate/client/monk_scripts/common** folder.

*Note:* *After you compose and save the required Monk script, you must select* **File, Commit to Sandbox** *from the e*Gate Schema Designer so that the script will be available for your use.*

Upon completing the Monk script and committing it to the Sandbox, you can create and configure the e*Way as follows:

1 In the components pane of the Schema Designer, select the Control Broker and click

   to add a new e*Way.

2 Enter **Receive_Mail** for the component name and click **OK**.

3 Select the newly created e*Way and click   to display the e*Way's properties.

4  Use the **Find** button to select **stcewgenericmonk.exe** as the executable file.

5  Click **New** to create a new configuration file.

6  Select **stcewemail** in the e*Way Template Selection window, and click **OK**.

7  Enter the parameters for the e*Way as shown in Table 5.

**Table 5**  Receive_Mail e*Way Parameters

| Parameter | Value |
|---|---|
| **General Settings** | Default |
| **Communication Setup** | |
| Exchange Data Interval | 20 |
| Zero Wait between successful Exchanges | Default |
| Start Exchange Data Schedule | blank |
| Stop Exchange Data Schedule | blank |
| Down Timeout | Default |
| Up Timeout | Default |
| Resend Timeout | Default |
| **Monk Configuration** | |
| Additional Path | Blank |
| Auxiliary Library Directories | monk_library/ewemail (default) |
| Monk Environment Initialization File | email-init.monk (default) |
| Startup Function | email-startup (default) |
| Process Outgoing Message Function | email-outgoing (default) |
| Exchange Data With External Function | email-exchange (default) |
| External Connection Establishment Function | email-extconnect (default) |
| External Connection Verification Function | email-verify (default) |
| External Connection Shutdown Function | email-shutdown (default) |
| Positive Acknowledgment Function | email-ack (default) |
| Negative Acknowledgment Function | email-nack (default) |
| Shutdown Command Notification Function | blank |
| Email Setup (must fill up the information for the following fields): | User Name: (Account to poll from)<br>Email Address: blank<br>Reply Address: blank<br>Outbound Server: blank<br>Outbound Port: blank<br>Inbound Server: (Name of inbound e-mail server)<br>Inbound Port: (non-conflicting port number) |

8  Select **Save** from the **File** menu. Enter **Receive_Mail** as the file name and click **Save**.

9  Select **Promote to Run Time** from the **File** menu. Click **OK** to promote the file.

10 A message will notify you that the file has been promoted to run time. Click **OK** to close the e*Way configuration file editor.

11 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

12 Click **OK** to save the e*Way properties.

## 4.2.3. Create and Configure the Event Types

The next step is creating the two Event Types previously mentioned, **Sort_Receive** and **Sort_Send**. These are simple Event Types that process e-Mail Events. **Sort_Send** identifies an e-mail that will be to forwarded to specified recipients, while **Sort_Receive** identifies an e-mail message that will be stored in a specified location.

There are two ways in which you can create an Event Type: open an existing Event Type Definition file and make modifications as needed, or create a new ETD file. (For more details about Event Type Definitions, see "To create Event Type Definitions" on page 34.

The following sections provide basic steps for creating each Event Type.

**To create the Sort_Send Event Type**

1 Click **Options** on the menu bar. Select Default Editor from the drop down menu. Two choices are available for Default Editor, Java or Monk. In this case select Monk.

2 Highlight the Event Types folder on the Components tab of the e*Gate Navigator.

3 On the Palette, click .

4 Enter **Sort_Send** as the name, then click **OK.**

5 Select **Sort_Send**, then click to edit its properties.

6 When the Properties window opens, click the **New** button.

When the ETD Editor opens, type sort_send as the file name, then click OK. Add the root node and the subnodes. When you are finished, the file should be similar to the following:

**Figure 11**   Sample ETD File for Sort_Send Event Type
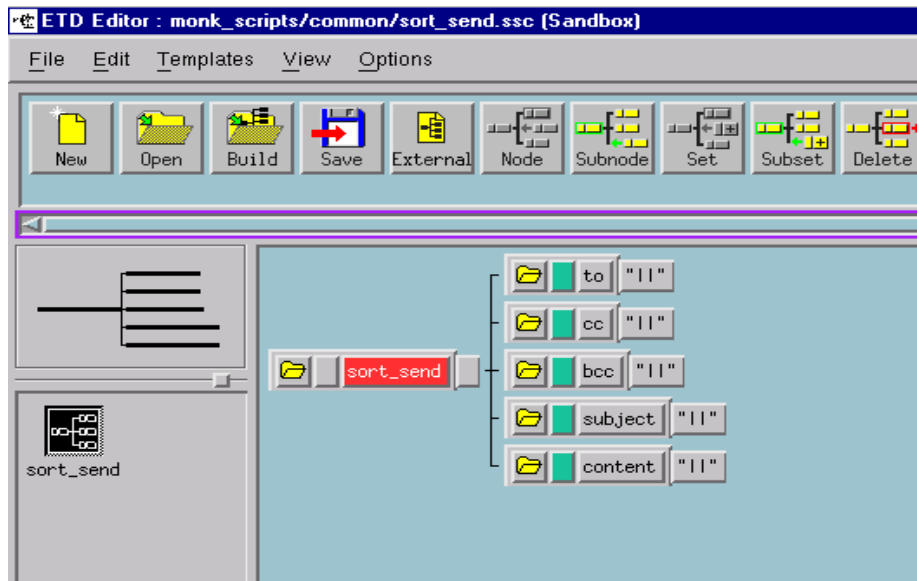


7   Save the file **sort_send.ssc**, and exit from the ETD Editor.

8   Click **OK** to save the Event Type properties.

**To create the Sort_Receive Event Type**

1   Highlight the Event Types folder on the Components tab of the e*Gate Navigator.

2   On the Palette, click [icon].

3   Enter **Sort_Receive** as the name, then click **OK.**

4   Select **Sort_Receive**, then click [icon] to edit its properties.

5   When the Properties window opens, click the **New** button.

When the ETD Editor opens, type sort_receive as the file name, then click OK. Add the root node and the subnodes. When you are finished, the file should be similar to the following:

**Figure 12** Sample ETD File for Sort_Receive Event Type
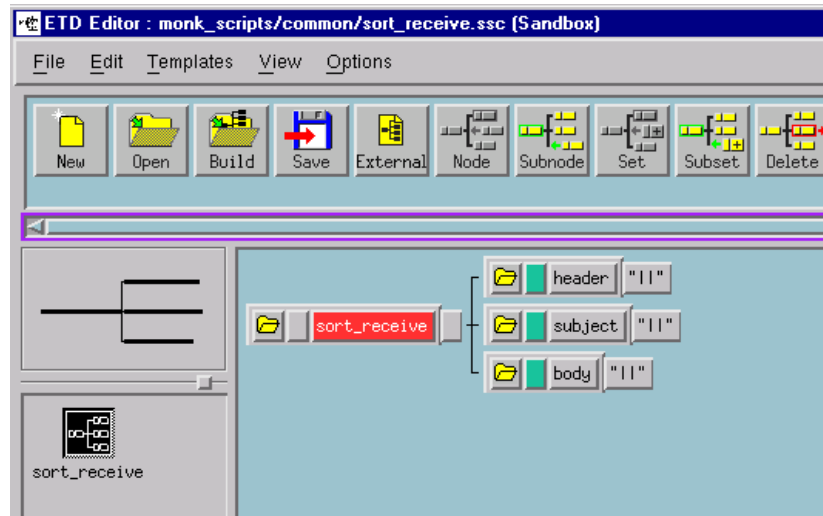


6  Save the file as **sort_receive.ssc,** and exit from the ETD Editor.

7  Click **OK** to save the Event Type properties.

## 4.2.4. Create the Collaboration Rules

The sample schema uses two Collaboration Rules: **crPass_Mail** and **crReceive_Mail**.

**To create the crPass_Mail Collaboration Rule**

*Note:* *This Collaboration Rule is associated with the **Sort_Send** Event Type, and is utilized to process outgoing e-mail.*

1  Select the Navigator's **Components** tab in the e*Gate Schema Designer.

2  In the Navigator, select the **Collaboration Rules** folder.

3  On the Palette, click

4  Enter **crPass_Mail** as the name of the new Collaboration Rule, then click **OK.**

5  Select **crPass_Mail**, then click to edit its properties.

6  In the **Service** field on the **General** tab, select **Pass Through** as the Collaboration Service.

7  On the **Subscriptions** tab, select **Sort_Send** as the required input Event Type.

8  On the **Publications** tab, select **Sort_Send** as the default output Event Type.

9  Click **OK.**

**To create the crReceive_Mail Collaboration Rule**

*Note:* *This Collaboration Rule is associated with the **Sort_Receive** Event Type, and is utilized to receive e-mail from an external e-mail account.*

1  Select the Navigator's **Components** tab in the e*Gate Schema Designer.

2  In the Navigator, select the **Collaboration Rules** folder.

3  On the Palette, click

4  Enter **crReceive_Mail** as the name of the new Collaboration Rule, then click **OK.**

5  Select **crReceive_Mail**, then click          to edit its properties.

6  In the **Service** field on the **General** tab, select **Pass Through** as the Collaboration Service.

7  On the **Subscriptions** tab, select **Sort_Receive** as the required input Event Type.

8  On the **Publications** tab, select **Sort_Receive** as the default output Event Type.

9  Click **OK**.

## 4.2.5. Add the Intelligent Queues (IQs)

This sample schema requires two IQs: **iq_Send** that processes e-mail for external recipients, and **iq_Receive** that processes external e-mail for storage in a specified location.

**To add the iq_Send and iq_Receive Intelligent Queues**

1  In the components pane of the Schema Designer, select the IQ Manager. Click          to create the new IQ.

2  Enter the name **iq_Send** and click **OK** to save the IQ.

3  Select the IQ Manager and click          to display the IQ Manager's properties.

4  On the **General** Tab, select the **STC_Standard** service. The default Event Type *Get Interval* of 100 Milliseconds is satisfactory.

5  On the **Advanced** tab, make sure that *Simple publish/subscribe* is checked under the **IQ behavior** section.

6  Click **OK**.

7  Repeat steps 1 through 6, but substitute **iq_Receive** as the name of the IQ.

## 4.2.6. Add and Configure the Collaborations

Each e*Way requires at least one Collaboration. This sample schema consists of four Collaborations, as follows:

- col_Inbound
- col_Receive_Mail
- col_Outbound
- col_Send_Mail

**To create the col_Inbound Collaboration**

*Note: This Collaboration is a member of the **Inbound e*Way**, and applies the **crPassMail** Collaboration Rule.*

1  In the components pane of the Schema Designer, select the **Inbound** e*Way.

2  Click the [icon] button to create a new Collaboration.

3  Enter the name **col_Inbound** and click **OK.**

4  Select the newly created Collaboration and click [icon] to display the Collaboration's properties.

5  Select **crPass_Mail** from the list of Collaboration Rules.

6  Click **Add** to add a new Subscription.

*Note: Define the input Event Type and source to which this Collaboration will subscribe.*

7  Select the **Sort_Send** Event Type and the **<EXTERNAL>** source.

8  Click **Add** to add a new Publication.

*Note: Define the input Event Type and destination to which this Collaboration will publish.*

9  Select the **Sort_Send** Event Type and the **iq_Send** destination.

10  Click **OK** to close the Collaboration's properties.

**To create the col_Receive_Mail Collaboration**

*Note: This Collaboration is a member of the **Receive_Mail** e*Way, and applies the **crReceive_Mail** Collaboration Rule.*

1  In the components pane of the Schema Designer, select the **Receive_Mail** e*Way.

2  Click the [icon] button to create a new Collaboration.

3  Enter the name **col_Receive_Mail** and click **OK.**

4   Select the newly created Collaboration and click ![icon] to display the Collaboration's properties.

5   Select **crReceive_Mail** from the list of Collaboration Rules.

6   Click **Add** to add a new Subscription.

*Note:*   *Define the input Event Type and source to which this Collaboration will subscribe.*

7   Select the **Sort_Receive** Event Type and the **<EXTERNAL>** source.

8   Click **Add** to add a new Publication.

*Note:*   *Define the input Event Type and destination to which this Collaboration will publish.*

9   Select the **Sort_Receive** Event Type and the **iq_receive** destination.

10   Click **OK** to close the Collaboration's properties.

**To create the col_Outbound Collaboration**

*Note:*   *This Collaboration is a member of the **Outbound** e\*Way, and applies the **crReceive_Mail** Collaboration Rule.*

1   In the components pane of the Schema Designer, select the **Outbound** e\*Way.

2   Click the ![icon] button to create a new Collaboration.

3   Enter the name **col_Outbound** and click **OK.**

4   Select the newly created Collaboration and click ![icon] to display the Collaboration's properties.

5   Select **crReceive_Mail** from the list of Collaboration Rules.

6   Click **Add** to add a new Subscription.

*Note:*   *Define the input Event Type and source to which this Collaboration will subscribe.*

7   Select the **Sort_Receive** Event Type and the **col_Receive_Mail** Collaboration source.

8   Click **Add** to add a new Publication.

*Note:*   *Define the input Event Type and destination to which this Collaboration will publish.*

9   Select the **Sort_Receive** Event Type and the **<EXTERNAL>** destination.

10   Click **OK** to close the Collaboration's properties.

**To create the col_Send_Mail Collaboration**

*Note:* *This Collaboration is a member of the **Send_Mail** e\*Way and applies the **crPass_Mail** Collaboration Rules.*

1 In the components pane of the Schema Designer, select the **Send_Mail** e\*Way.

2 Click the ![icon] button to create a new Collaboration.

3 Enter the name **col_Send-Mail** and click **OK.**

4 Select the newly created Collaboration and click ![icon] to display the Collaboration's properties.

5 Select **crPass_Mail** from the list of Collaboration Rules.

6 Click **Add** to add a new Subscription.

*Note:* *Define the input Event Type and source to which this Collaboration will subscribe.*

7 Select the **Sort_Send** Event Type and the **col_Inbound** source.

8 Click **Add** to add a new Publication.

*Note:* *Define the input Event Type and destination to which this Collaboration will publish.*

9 Select the **Sort_Send** Event Type and the **<EXTERNAL>** destination.

10 Click **OK** to close the Collaboration's properties.

## 4.2.7. Run the Schema

Start the Control Broker from a command prompt to execute the e-Mail Test schema.

**To run the Control Broker**

1 From a command line, type the following command:

```
stccb -ln logical_name -rh registry -rs schema -un user_name -up
password
```

*logical_name* is the logical name of the Control Broker

*registry* is the name of the Registry Host.

**schema** is the name of the Registry Schema, and

*user_name and password* are a valid e\*Gate username/password combination.

2 Exit (*for Windows*) from the command line prompt, and start the Schema Manager GUI.

3 When prompted, specify the hostname which contains the control broker you started in Step 1 above.

4 Select the **e-Mail_Test** schema.

5 After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, **hostname_igmgr**, then click the right button of the mouse, and select **Start**. Highlight each of the e*Ways, right click the mouse, and select **Start**.

## 4.3 Sample Monk Scripts

This section provides examples of Monk scripts for use in the sample implementation in this chapter. These files are for demonstration purposes only, and are intended to show the requirements for an e*Way that sends or receives e-mail, to or from an external source. If you copy this file, some substitutions are required.

## 4.3.1. Sample Monk File: email-outgoing.monk

This Monk file maps the e-Mail account, from which you are sending mail, to the Event Type that forwards the e-mail.

```
;;External Monk Function
;;usage:(email-outgoing event-string)

;;
;;External Monk Function
;;usage:(http-outgoing event-string)

;: First define a structure to separate the inbound event into url,
request and content

;:- Delimiter Structure for parsing the url, request and content
(define email-delm '(
 ("||")
 ))

;:- MsgStructure Definition for parsing the url, request and content

(define email-struct ($resolve-event-definition (quote
  (email ON 1 1 und und und -1
    (to ON 1 1 und und und -1);:= {0.0:N}
    (cc ON 1 1 und und und -1);:= {0.1:N}
    (bcc ON 1 1 und und und -1);:= {0.2:N}
    (subject ON 1 1 und und und -1);:= {0.3:N}
    (content ON 1 1 und und und -1);:= {0.4:N}
  ) ;:= {0:N}
)))

(define email-outgoing
   (let ((input (make-message-structure email-delm email-struct))
       (to "") (cc "") (bcc "") (subject "") (content "")
     )
    (lambda (message-string)
           (let
             ( (return-value "") )     ; Default to failure

           (display "Inside email-outgoing\n")
           ;;
```

```
                         ;; First we will parse our event structure
                         ;;
                         (message-parse input message-string)
                         (set! to (get ~input%email.to))
                         (set! cc (get ~input%email.cc))
                         (set! bcc (get ~input%email.bcc))
                         (set! subject (get ~input%email.subject))
                         (set! content (get ~input%email.content))

                         ;;
                         ;; Now create and send our message
                         ;;
                         (define hRet (email-create-mailretrieve-handle))

                         (define hMail (email-new-message))
                         (email-set-mailhost hMail "atlas")
                         (email-set-mailhost-port hMail 25)
                         (email-set-login-name hMail "ewaytest")
                         (email-set-password hMail "stcd23")
                         (email-set-sender-name hMail "Email e*Way")
                         (email-set-sender-address hMail "ewaytest@stc.com")
                         (email-set-reply-address hMail "ewaytestreply@stc.com")

                         (email-add-recipient hMail to)
                         (email-add-cc-recipient hMail cc)
                         ;(email-add-bcc-recipient hMail bcc)
                         (email-set-subject hMail subject)
                         (email-set-content hMail content)
                         (define sent (email-send hMail))
          (display to)
          (display "---\n")
          (display cc)
          (display "---\n")
          (display bcc)
          (display "---\n")
          (display subject)
          (display "---\n")
          (display content)
          (display "---\n")
          (display sent)
          (display "---\n")
                                    ;return-value
                         ""

              ) ; let

          );lambda
        );let
);define email-outgoing
```

## 4.3.2. Sample Monk file: email-exchange.monk

This Monk file is used for polling an e-Mail account. Enter the password for the account you are polling, and update the script so that it is valid for the ETD you are using.

```
;;External Monk Function
;;usage:(email-exchange)

(define email-exchange
    (lambda ()
        (let
            ( (retData "") )        ; Default to empty
```

```
        (if (string? E-MAIL_CONFIGURATION_INBOUNDPORT)
            (define E-MAIL_CONFIGURATION_INBOUNDPORT (string->number
    E-MAIL_CONFIGURATION_INBOUNDPORT))
        )

        ; Get the next message from our mail server
        ;(display "Inside *** email-exchange\n")
        ;(define hRet (email-create-mailretrieve-handle))
        ;(email-set-mailhost hRet E-
    MAIL_CONFIGURATION_INBOUNDSERVER)
        ;(email-set-mailhost-port hRet E-
    MAIL_CONFIGURATION_INBOUNDPORT)
        ;(email-set-login-name hRet E-MAIL_CONFIGURATION_USERNAME)
            ;(email-set-password hMail E-
    MAIL_CONFIGURATION_ENCRYPTED_PASSWORD)
        ;(email-retrieve-mail hRet)
        ;(define hHeader (email-get-message-header hRet))
        ;(set! retData (email-get-message-content hRet))

        ;(set! attachmentCount (email-get-attachment-count hRet))
        ;(define pszFileName (email-save-attachment hRet 0 ""))

            retData

        ) ; let
    );lambda
);define
```

## 4.4   Possible Order for API calls

The following examples are presented as the suggested order of API calls for both a send and receive scenario.

### 4.4.1.  Send

1   **email-new-message** on page 76

2   **email-set-mailhost** on page 87

3   **email-set-mailhost-port** on page 88

4   **email-set-login-name** on page 90

5   **email-set-sender-name** on page 81

6   **email-set-sender-address** on page 82

7   **email-set-reply-address** on page 83

8   **email-add-recipient** on page 77 (must call at least once)

9   **email-add-cc-recipient** on page 78 (optional)

10   **email-add-bcc-recipient** on page 79 (optional)

11   **email-set-subject** on page 84

12   **email-set-content** on page 85

13 **email-add-attachment** on page 86 (optional, multiple)

14 **email-send** on page 98

## 4.4.2. Receive

1 **email-create-mailretrieve-handle** on page 89

2 **email-set-mailhost-port** on page 88

3 **email-set-mailhost** on page 87

4 **email-set-login-name** on page 90

5 **email-set-password** on page 91

6 **email-retrieve-mail** on page 92

7 **email-get-message-header** on page 94 (contains complete email header information)

8 **email-get-message-content** on page 95 (contains complete text content)

9 **email-get-attachment-count** on page 96

10 **email-save-attachment** on page 97

11 **email-release-mailretrieve-handle** on page 93

# e-Mail e*Way Functions

The e-Mail e*Way's functions fall into the following categories:

- **Basic Functions** on page 56
- **e-Mail e*Way Standard Functions** on page 64
- **e-Mail e*Way Native Functions** on page 75

## 5.1 Basic Functions

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

*Note:* *The functions described in this section 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 basic functions are

**start-schedule** on page 57

**stop-schedule** on page 58

**send-external-up** on page 59

**send-external-down** on page 60

**get-logical-name** on page 61

**event-send-to-egate** on page 62

**shutdown-request** on page 63

## 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 affect any defined schedules.

**Parameters**

None.

**Return Values**

None.

**Throws**

None.

## stop-schedule

**Syntax**

```
(stop-schedule)
```

**Description**

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

**Parameters**

None.

**Return Values**

None.

**Throws**

None.

## 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.

## 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.

# 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 Schema Designer).

**Throws**

None.

# event-send-to-egate

**Syntax**

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

**Description**

**event-send-to-egate** sends data that the e*Way has already received from the external system into the e*Gate system as an Event.

**Parameters**

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

**Return Values**

**Boolean**
Returns **#t** (true) if the data is sent successfully; otherwise, returns **#f** (false).

**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.

## 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 30**) Once this function is called, the shutdown proceeds immediately.

Once interrupted, the e*Way's shutdown cannot proceed until this Monk function is called. If you do interrupt an e*Way shutdown, we recommend that you complete the process in a timely fashion.

**Parameters**

None.

**Return Values**

None.

**Throws**

None.

## 5.2 e-Mail e*Way Standard Functions

The functions in this section control the e*Way's communications center and are defined within the configuration file.

The current suite of Standard functions are:

**email-ack** on page 65

**email-exchange** on page 66

**email-extconnect** on page 67

**email-init** on page 68

**email-nack** on page 69

**email-notify** on page 70

**email-outgoing** on page 71

**email-shutdown** on page 72

**email-startup** on page 73

**email-verify** on page 74

## email-ack

**Syntax**

```
(email-ack arg)
```

**Description**

**email-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 |
|------|------|-------------|
| arg | 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.

## email-exchange

**Syntax**

```
(email-exchange)
```

**Description**

**email-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 string, containing Event data, indicates successful operation, and the returned 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 re-executed with the same input Event.

**Throws**

None.

**Additional Information**

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

## email-extconnect

**Syntax**

```
(email-extconnect)
```

**Description**

**email-extconnect** 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.

## email-init

**Syntax**

```
(email-init)
```

**Description**

**email-init** begins the initialization process for the e*Way. This function loads the **stc_monkemail.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.

## email-nack

**Syntax**

```
(email-nack arg)
```

**Description**

**email-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 |
|------|------|-------------|
| arg | 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.

## email-notify

**Syntax**

```
(email-notify command)
```

**Description**

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

**Parameters**

| Name | Type | Description |
| --- | --- | --- |
| command | string | The function that passes the string "SHUTDOWN_NOTIFICATION" to the external system before the e*Way shuts down. |

**Return Values**

**string**
Returns a null string.

**Throws**

None.

**Additional Information**

See **"Shutdown Command Notification Function" on page 24** for more information.

## email-outgoing

### Syntax

```
(email-outgoing event-string)
```

### Description

**email-outgoing** sends a received Event 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 re-executed with the same input Event.

"DATAERR" indicates the function had a problem processing data. If the e*Gate journal is enabled, the Event is journaled and the failed Event count is increased. (The input Event is essentially skipped in this process.) Use the **event-send-to-egate** function to place bad events in a bad event queue. See **event-send-to-egate** on page 62 for more information on this function.

### Throws

None.

### Additional Information

See **Process Outgoing Message Function** on page 26 for more information.

## email-shutdown

**Syntax**

```
(email-shutdown shutdown)
```

**Description**

**email-shutdown** requests that the external connection shut down. A return value of "SUCCESS" indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a (**shutdown-request** on page 63) call from within a Monk function to allow the requested shutdown to process to continue.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| shutdown | string | The function that passes the string "SUSPEND_NOTIFICATION" to the external system before the e*Way shuts down. |

**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** for more information.

## email-startup

**Syntax**

```
(email-startup)
```

**Description**

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

**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.

# email-verify

**Syntax**

```
(email-verify)
```

**Description**

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

**Parameters**

None.

**Return Values**

**string**

"UP" if connection established. Any 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  e-Mail e*Way Native Functions

The following functions are APIs native to the e-Mail e*Way that enable the e*Way to send and retrieve e-mail by accessing the specified mail server.

*Note:*  *The functions described in this section can only be called from within a Collaboration Rules script.*

The functions are:

## email-new-message

**Syntax**

```
(email-new-message)
```

**Description**

**email-new-message** returns the handle for the new message.

**Parameters**

None.

**Return Values**

**handle**
The handle associated with the new e-mail message.

**Throws**

None.

## email-add-recipient

**Syntax**

```
(email-add-recipient hMsg pszRecipient)
```

**Description**

**email-add-recipient** establishes the e-mail address of the recipient.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszRecipient | string | The e-mail address of the recipient. |

**Return Values**

**Boolean**

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

**Throws**

None.

**Additional Information**

To add more than one recipient, the function must be repeated for each addition.

## email-add-cc-recipient

**Syntax**

```
(email-add-cc-recipient hMsg pszRecipient)
```

**Description**

**email-add-cc-recipient** specifies the e-mail addresses of the recipient to receive a copy of the e-mail.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszRecipient | string | The e-mail address of the copied recipient(s). |

**Return Values**

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

**Throws**

None.

**Additional Information**

To add more than one recipient, the function must be repeated for each addition.

## email-add-bcc-recipient

**Syntax**

```
(email-add-bcc-recipient hMsg pszRecipient)
```

**Description**

**email-add-bcc-recipient** specifies the address of the recipient to receive a blind copy of the message. A blind copy is one where the address of the recipient copied does not appear on the message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszRecipient | string | The e-mail address of the recipient to receive the blind copy of the message. |

**Return Values**

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

**Throws**

None.

**Additional Information**

To add more than one recipient, the function must be repeated for each desired addition.

## email-add-header

**Syntax**

```
(email-add-header hMsg field_name field_value)
```

**Description**

**email-add-header** adds a header to an outgoing eMail message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| field_name | string | The name of the field in the header. |
| filed_value | string | The value of the field in the header. |

**Return Values**

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

**Throws**

None.

**Additional Information**

The format of the header in the eMail is *field-name*:*field-value*. **email-new-message** must be called before this function. **email-add-header** uses the handle returned to determine the correct message to add the header. **email-add-header** must then be executed before **email-send** is called. To add additional headers, this function must be repeated for each desired header.

**Examples**

```
(define hMsg (email-new-message))
(if (email-add-header hMsg "Content-Type" "text/html")
  (display "Content-Type header added successfully!\n")
  (display "Unable to add Content-Type header!\n")
```

*Note:  Do not add a date header with a get-timestamp.*

## email-set-sender-name

**Syntax**

```
(email-set-sender-name hMsg pszSenderName)
```

**Description**

**email-set-sender-name** attaches the sender's name to the message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszSenderName | string | The name of the sender of the message. |

**Return Values**

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

**Throws**

None.

## email-set-sender-address

**Syntax**

```
(email-set-sender-address hMsg pszSenderAddress)
```

**Description**

**email-set-sender-address** attaches the sender's address to the message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszSenderAddress | string | The e-mail address of the sender. |

**Return Values**

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

**Throws**

None.

## email-set-reply-address

### Syntax

```
(email-set-reply-address hMsg pszSenderAddress)
```

### Description

**email-set-reply-address** specifies the return address of the e-mail message.

### Parameters

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszSenderAddress | string | The e-mail address of the sender. |

### Return Values

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

### Throws

None.

## email-set-subject

**Syntax**

```
(email-set-subject hMsg pszSubject)
```

**Description**

**email-set-subject** specifies the text to appear on the subject line of the message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque | The handle returned by **email-new-message**. |
| pszSubject | string | The string to appear on the subject line of the message. |

**Return Values**

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

**Throws**

None.

## email-set-content

**Syntax**

```
(email-set-content hMsg pszContent)
```

**Description**

**email-set-content** specifies the content of the message being sent.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszContent | string | The content of the message. |

**Return Values**

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

**Throws**

None.

## email-add-attachment

**Syntax**

```
(email-add-attachment hMsg pszFileLocation)
```

**Description**

**email-add-attachment** specifies the absolute path location of the file attachment.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszFileLocation | string | An absolute path location to the attachment file. |

**Return Values**

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

**Throws**

None.

## email-set-mailhost

**Syntax**

```
(email-set-mailhost hMsg pszMailHost)
```

**Description**

**email-set-mailhost** specifies the host name of the SMTP server.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszMailHost | string | The host name of the SMTP server. |

**Return Values**

**Boolean**

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

**Throws**

None.

## email-set-mailhost-port

**Syntax**

```
(email-set-mailhost-port hMsg iPortNumber)
```

**Description**

**email-set-mailhost-port** specifies the port number associated with the SMTP server.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| iPortNumber | integer | The port number associated with the SMTP server. |

**Return Values**

**Boolean**

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

**Throws**

None.

## email-create-mailretrieve-handle

**Syntax**

```
(email-create-mailretrieve-handle)
```

**Description**

**email-create-mailretrieve-handle** specifies the connection handle for mail retrieval.

**Parameters**

None.

**Return Values**

**handle**
The handle needed to access the mail retrieval object.

**Throws**

None.

## email-set-login-name

**Syntax**

```
(email-set-login-name hMsg pszLoginName)
```

**Description**

**email-set-login-name** specifies the login name for the user's e-mail account.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszLoginName | string | The user id for the e-mail account being accessed on the server. |

**Return Values**

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

**Throws**

None.

## email-set-password

**Syntax**

```
(email-set-password hMsg pszPassword)
```

**Description**

**email-set-password** specifies the password associated with the login name specified by **email-set-login-name**.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| pszPassword | string | The password associated with the login name specified by **e-mail-set-login-name**. |

**Return Values**

**Boolean**

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

**Throws**

None.

## email-retrieve-mail

**Syntax**

```
(email-retrieve-mail hMsg)
```

**Description**

**email-retrieve-mail** performs the mail retrieval.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-create-mailretrieve-handle**. |

**Return Values**

**Boolean**
Returns **#t** (true) if mail is retrieved successfully and stored in the handle created by **email-create-mailretrieve-handle**; otherwise, returns **#f** (false) when the server is successfully contacted but there is no mail to retrieve.

**Throws**

Monk exception if a connection error occurs.

**Additional Information**

Before calling this function:

1 Create the message handle.

2 Set all necessary fields

   A  Userid

   B  Password

   C  Port

3 Once all desired fields are set, call **email-retrieve-mail**.

# email-release-mailretrieve-handle

**Syntax**

```
(email-release-mailretrieve-handle hMsg)
```

**Description**

**email-release-mailretrieve-handle** terminates the connection and releases the memory associated with the specified handle.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |

**Return Values**

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

**Throws**

None.

## email-get-message-header

**Syntax**

```
(email-get-message-header hMsg)
```

**Description**

**email-get-message-header** returns a string containing the header from the specified message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |

**Return Values**

**string**
Returns a string that contains the e-mail header.

**Throws**

None.

# email-get-message-content

**Syntax**

```
(email-get-message-content hMsg)
```

**Description**

**email-get-message-content** retrieves the content of the message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |

**Return Values**

**string**

Returns the text content of the message.

**Throws**

None.

## email-get-attachment-count

**Syntax**

```
(email-get-attachment-count hMsg)
```

**Description**

**email-get-attachment-count** returns the number of attachments to the message.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |

**Return Values**

**integer**
Returns an integer indicating the number of attachments.

**Throws**

None.

# email-save-attachment

**Syntax**

```
(email-save-attachment hMsg AttNumber AttchmentName)
```

**Description**

**email-save-attachment** saves the attachment to the "egate\client\received_email" directory.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |
| AttNumber | integer | The index of the attachment to save. The index is zero based. The maximum valid number is the value returned by email-get-attachmentcount - 1. |
| AttachmentName | string | The absolute path to which the attachment should be saved. |

**Return Values**

**string**
Returns filename and absolute path associated with the attachment if successful; otherwise, returns **#f** (false).

**Throws**

None.

**Additional Information**

In this function the **AttachmentName** parameter must be specified. If an empty string ("") is specified, the file will be saved with the filename of the attachment in the egate\client\received_email directory.

If a file currently exists with the same name, or if the attachment did not have a file name sent with it, the function will define a filename of "**attachmentx**" where "**x**" acts as a numeric identifier.

*Note:* *The E-mail e*Way does not support sending/receiving messages containing multi level embedded MIME parts.*

## email-send

**Syntax**

(email-send *hMsg)*

**Description**

**email-send** sends the specified message to the port host specified in
**email-set-mailhost-port**.

**Parameters**

| Name | Type | Description |
|------|------|-------------|
| hMsg | opaque handle | The handle returned by **email-new-message**. |

**Return Values**

**Boolean**
Returns **#t** (true) if successful; otherwise, returns **#f** (false).

**Throws**

None.

**Additional Information**
The SNMP size limitation of 1024 characters limits the possible size of the outgoing
message.

Use full tracing capabilities to access mail server error messages for debugging
purposes.

# Index

## O

OutboundPort **32**
OutboundServer **32**
overview **6**

## P

parameters **12**
Positive Acknowledgment Function parameter **29**
Process Outgoing Message Function parameter **26**

## R

receive scenario **55**
ReplyAddress **32**
Resend Timeout parameter **16**

## S

sample e-Mail schema **39**
Sample Monk File
    email-outgoing.monk **52**
sample schema
    creating **37**
    sample Monk scripts **52**
    starting the Control Broker **51**
send scenario **54**
send-external-down function **60**
send-external-up function **59**
Shutdown Command Notification Function
parameter **30**
shutdown-request **63**
standard functions **64**
    email-ack **65**
    email-exchange **66**
    email-extconnect **67**
    email-init **68**
    email-nack **69**
    email-notify **70**
    email-outgoing **71**
    email-shutdown **72**
    email-startup **73**
    email-verify **74**
start-schedule function **57**
Startup Function parameter **26**
Stop Exchange Data Schedule parameter **16**
stop-schedule function **58**
supported operating systems **7**
system requirements **7**

## U

Up Timeout parameter **16**

Username **31**

## Z

Zero Wait Between Successful Exchanges parameter **14**