SeeBeyond ICAN Suite

e*Way Intelligent Adapter for MSMQ User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Monk Version

Version 20050406012257.

Contents

Chapter 1

Introduction

This guide explains the SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way™ Intelligent Adapter for e*Way Intelligent Adapter for MSMQ (Microsoft Message Queue Server). This chapter provides an introduction to the guide and the e*Way.

1.1 Overview

The MSMQ e*Way enables the e*Gate system to integrate business event delivery between applications over various networks. This allows messages or data to be sent over multiple operating systems and networking protocols by insulating the application developer from the details of the various operating systems and network interfaces.

1.1.1 Intended Reader

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

1.1.2 Components

The following components comprise the MSMQ e*Way:

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

A complete list of installed files appears in **Table 1 on page 9**.

1.2 Supported Operating Systems

The MSMQ e*Way is supported on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003

1.3 System Requirements

To use the MSMQ e*Way, you need to meet the following requirements:

- An e*Gate Participating Host
- 4 MB free disk space

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

- A TCP/IP network connection

The client components of MSMQ have their own requirements; see that system's documentation for details.

1.4 External System Requirements

To enable the e*Way to communicate properly with the MSMQ system, the following are required:

- Microsoft Messaging Queues, version 1.0 or 2.0
- SQL server version 6.5 or 7.0

Chapter 2

Installation

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

2.1 Windows Systems

2.1.1 Pre-installation

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

2.1.2 Installation Procedure

To install the MSMQ e*Way on Windows systems

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 Component 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 **MSMQ** and click **OK**.

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

13 You will return to the e*Way's 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** user guide.*

2.2 Files/Directories Created by the Installation

The MSMQ 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
bin\	stc_monkmsmq.dll
configs\stcgenericmonk	stcewmsmq.def
monk_library	msmq.gui
monk_library\ewmsmq	msmq-startup.monk msmq-shutdown.monk msmq-pos-ack.monk msmq-outgoing.monk msmq-neg-ack.monk msmq-init.monk msmq-incoming.monk msmq-conn-verify.monk msmq-conn-shutdown.monk msmq-conn-establish.monk

Chapter 3

Configuration

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

3.1 e*Way Configuration Parameters

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

To change e*Way configuration parameters:

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

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

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

- General Settings
- Communication Setup
- Monk 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 11** 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. When this maximum is reached, the message is considered "Failed" and is written to the journal file.

Required Values

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

Max Failed Messages

Description

Specifies the maximum number of failed messages (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

Selects whether error messages that begin with the string "DATAERR" that are received from the external system will be queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 25** for more information.

Required Values

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

See **"Schedule-driven Data Exchange Functions" on page 19** for information about how the e*Way uses this function.

3.1.2 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.*

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

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

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

- Exchange Data With External Function
- 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 25**, **"Exchange Data Interval" on page 13**, and **"Stop Exchange Data Schedule" on page 12** for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

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

Exchange Data Interval

Description

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

Required Values

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

Additional Information

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

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

See **"Down Timeout" on page 13** and **"Stop Exchange Data Schedule" on page 12** for more information about the data-exchange schedule.

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 26** 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 26** 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.

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**.

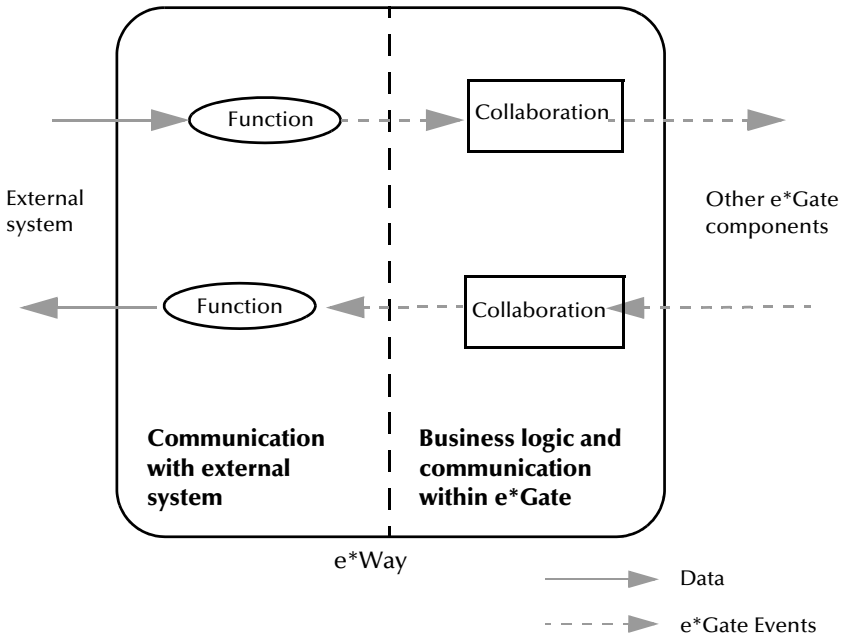
See **"Exchange Data with External Function" on page 25** for more information.

3.1.3 Monk Configuration

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

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 1 below) handles communication with the external system; the other half manages the Collaborations that process data and subscribe or publish to other e*Gate components.

Figure 1 e*Way internal architecture



The "communications half" of the e*Way uses Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events" and send those Events to Collaborations, and manage the connection between the e*Way and the external system. The **Monk Configuration** options discussed in this section control the Monk environment and define the Monk functions used to perform these basic e*Way operations. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as **Notepad**, or UNIX **vi**).

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

Operational Details

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

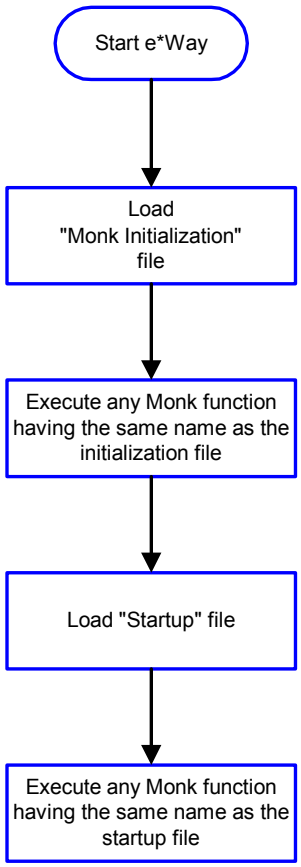
Type of Operation	Name
Initialization	**Startup Function** on page 24 (also see **Monk Environment Initialization File** on page 23)
Connection	**External Connection Establishment Function** on page 26 **External Connection Verification Function** on page 26 **External Connection Shutdown Function**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

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

Initialization Functions

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

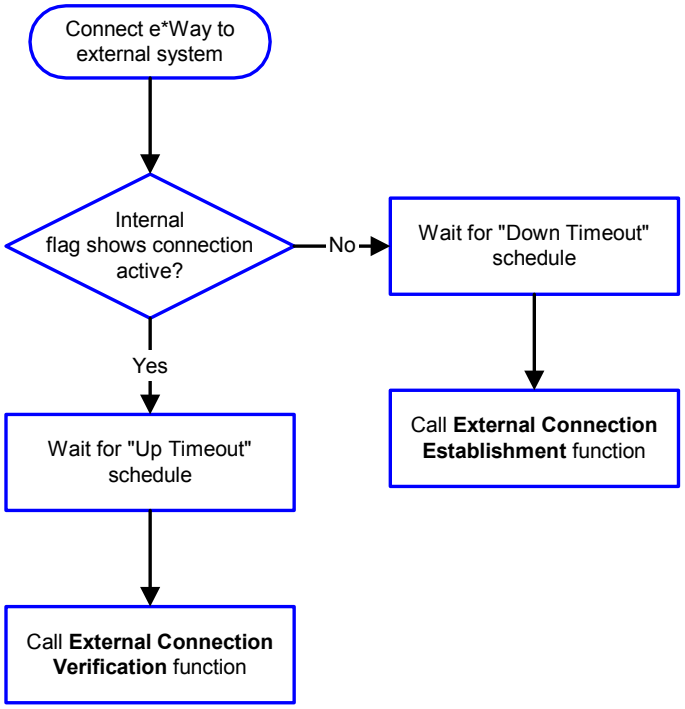
Figure 2 Initialization Functions



Connection Functions

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

Figure 3 Connection establishment and verification functions

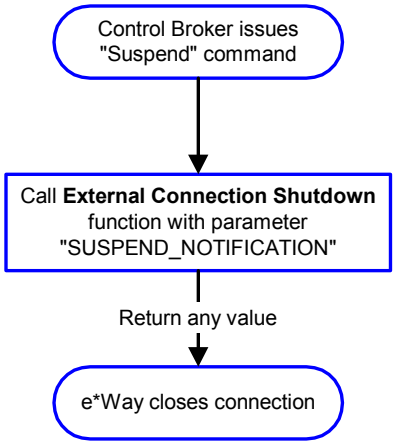


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 5 on page 20** *and* **Figure 7 on page 22** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 43 and* **send-external-down** *on page 44 for more information.*

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

Figure 4 Connection shutdown function



Schedule-driven Data Exchange Functions

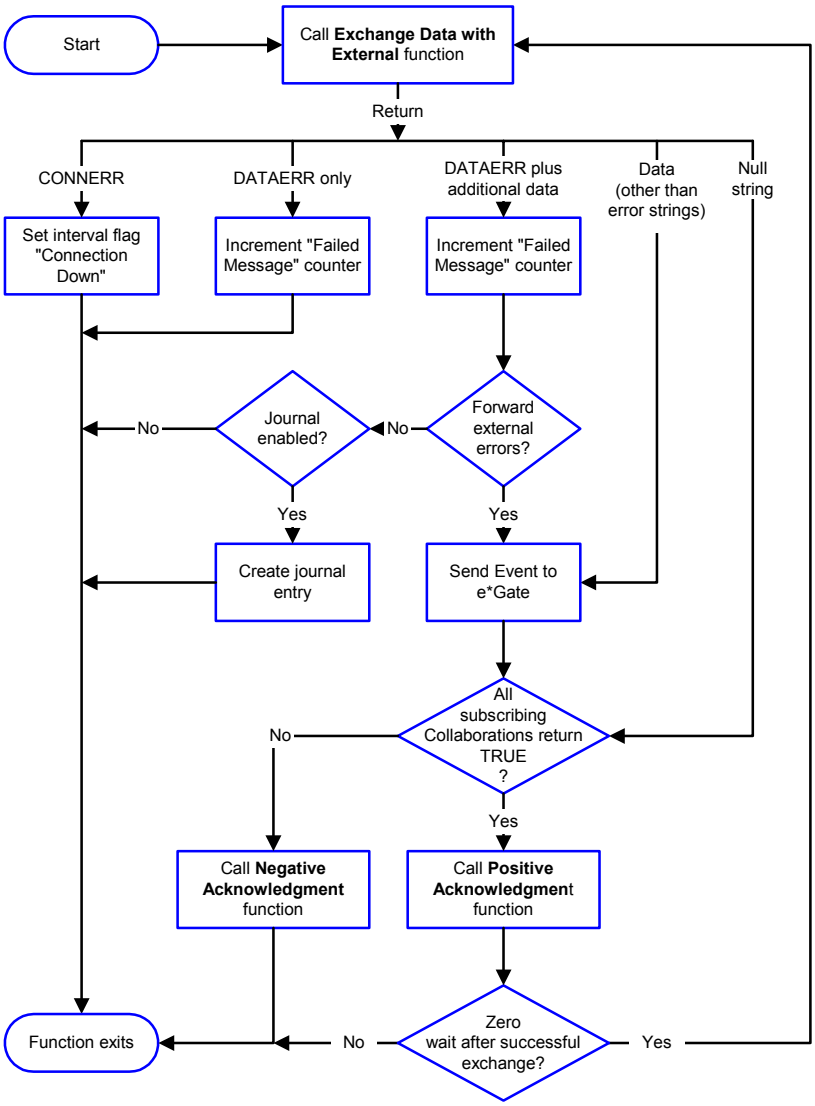
Figure 5 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgment Function** and **Negative Acknowledgment Function** are also called during this process.

"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.

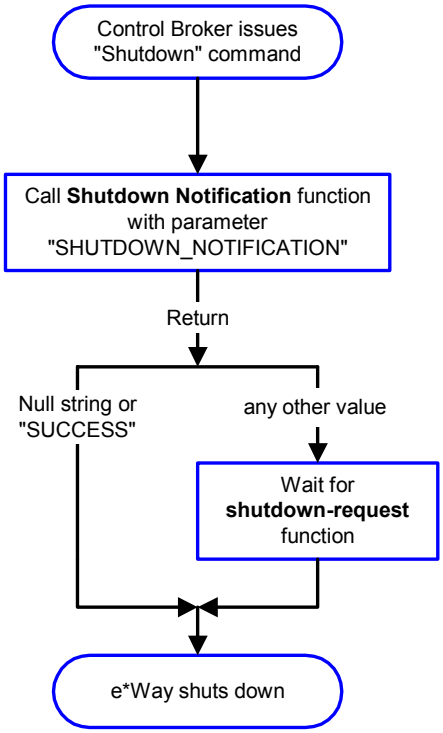
Figure 5 Schedule-driven data exchange functions



Shutdown Functions

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

Figure 6 Shutdown Functions



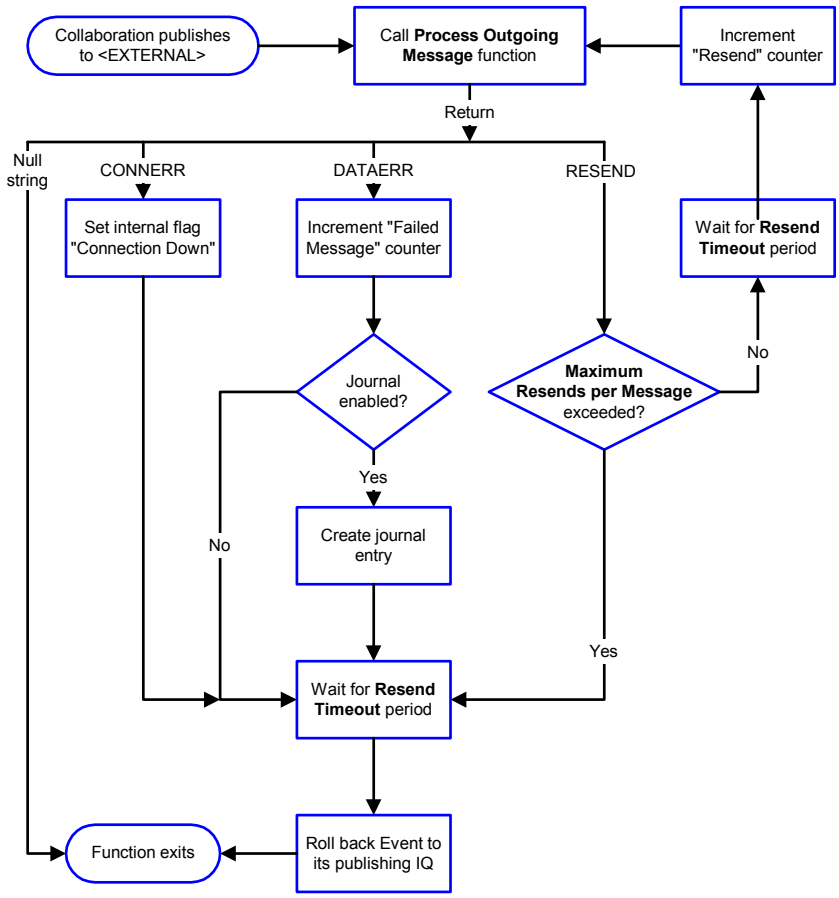
Event-driven Data Exchange Functions

Figure 7 on the next page illustrates event-driven data-exchange using the **Process Outgoing Message Function**.

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

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

Figure 7 Event-driven data-exchange functions



How to Specify Function Names or File Names

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

- .monk
- .tsc
- .dsc

Additional Path

Description

Specifies a path to be 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 "file selection" button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths. For example:

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

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

Auxiliary Library Directories

Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories will automatically be loaded into the e*Way's Monk environment. This parameter is optional and may be left blank.

Required Values

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

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.

This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which will be loaded after the auxiliary library directories are loaded. 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 path information is specified, that path will be appended to the "load

path." See **"Additional Path" on page 22** for more information about the "load path."(The default is **msmq-init** on page 48.)

Additional information

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

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

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 2 on page 17**).

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. This parameter is optional and may be left blank. (The default is **msmq-startup** on page 53.)

Additional information

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

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

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

The e*Way will load this file and try to invoke a function of the same base name as the file name (see **Figure 2 on page 17**). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

Process Outgoing Message Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***You may not leave this field blank.*** (The default is **msmq-outgoing** on page 50.)

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 7 on page 22** 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 44 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. This parameter is optional and may be left blank. (The default is **msmq-incoming** on page 48.)

Additional Information

The function accepts no input and must return a string (see **Figure 5 on page 20** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information will be sent into the e*Gate system.
- "CONNERR": Indicates that a problem with the connection to the external system has occurred.
- "DATAERR": Indicates that a problem with the data itself has occurred. The e*Way handles the string "DATAERR" and "DATAERR" plus additional data differently; see **Figure 5 on page 20** for more details.

- Any other string: The contents of the string are packaged as an inbound Event. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

This function is initially triggered by the **Start Data Exchange** schedule or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been ACKed or NAKed (by the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter. If this parameter is set to **Yes**, the e*Way will immediately call the **Exchange Data with External** function again; otherwise, the e*Way will not call the function until the next scheduled "start exchange" time or the schedule is manually invoked using the Monk function **start-schedule** (see **start-schedule** on page 42 for more information).

External Connection Establishment Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***This field cannot be left blank.*** (The default is **msmq-conn-establish** on page 46.)

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 **msmq-conn-verify** on page 47.)

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. This parameter is optional. (The default is **msmq-shutdown** on page 52.)

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 **msmq-pos-ack** on page 51.)

Additional Information

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

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the Positive Acknowledgment function will be called again, with the same input data.

- Null string: The function completed execution successfully.

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

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 **msmq-neg-ack** on page 49.)

Additional Information

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

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

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

Shutdown Command Notification Function

Description

Specifies a Monk function that will be called when the e*Way receives a "shut down" command from the Control Broker. This parameter is optional.

Required Values

The name of a Monk function. (The default is **msmq-shutdown** on page 52.)

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 45).

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 MSMQ Settings

This section contains the parameters the control the settings associated with the MSMQ connection.

GetQueue

Description

Specifies the Global Unique Identifier (GUID) for the queue from which to retrieve data.

Required Values

A string; this is a required field for inbound processing. The format must not begin with "Public," for example,

```
274059E7-6A46-11D3-B5A0-0004A69035BE
```

The Public string must be set in the **startup.monk** function as per the Monk sample scripts in **monklibrary/msmq**.

Additional Information

To obtain this information:

1. Access the MSMQ server.
2. Open the MSMQ explorer.
3. Select the queue for which the GUID is required.
4. View its properties.

For more information, see the MSMQ documentation.

PutQueue

Description

Specifies the queue to which the specified data is directed.

Required Values

A string. This field is required for outbound processing. The format must not begin with "Direct=OS". For example,

```
myservername\myqueue
```

The Direct=OS string must be appended in the Monk functions, as per the Monk sample scripts in the monklibrary/msmq.

Note: *The above example pertains to use with the GUI. If hard-coded into a script either format is acceptable.*

Additional Information

myservername: the name of the MSMQ server

myqueue: the name of the valid queue

For more information, please review the *MSMQ* documentation for further information.

GetTimeout

Description

Specifies the maximum number of milli-seconds the get request will wait before returning a no-data failure.

Required Values

An integer between 1 and 999999. The default is 500.

GetTransType

Description

Specifies the transaction processing type for all get operations.

Required Values

An integer. The possible values are:

0 = no transaction support
3 = part of single message transaction

PutTransType

Description

Specifies the transaction processing type for all put operations.

Required Values

An integer. The possible values are:

0 = no transaction support
3 = part of single message transaction

3.2 External Configuration Requirements

If a valid queue is not already present on the MSMQ server, a queue must be created. To do this:

1 Select the server.

2 Right click, select create a queue.

Note: *For more information, please review the MSMQ documentation for further information.*

Chapter 4

Implementation

This chapter discusses how to implement the MSMQ e*Way in a production environment.

4.1 Implementation Process: Overview

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

- Define Event Type Definitions (ETDs) to package the data being exchanged with the external system.
- In the e*Gate Schema Designer, do the following:
  - Define Event Types for the data.
  - Define Collaboration Rules to process Event data.
  - Define any IQs to which Event data will be published prior to sending it to the external system.
  - Create a new e*Way component and configure its properties.
  - Within the e*Way component, configure Collaborations to apply the required Collaboration Rules.
- Use the e*Way Editor to set the e*Way's configuration parameters.
- Be sure that any other e*Gate components are configured as necessary to complete the schema.
- Test the schema and make any necessary corrections.

See **"Creating the Sample Schema" on page 33** for examples of how the above steps are combined to create a working implementation.

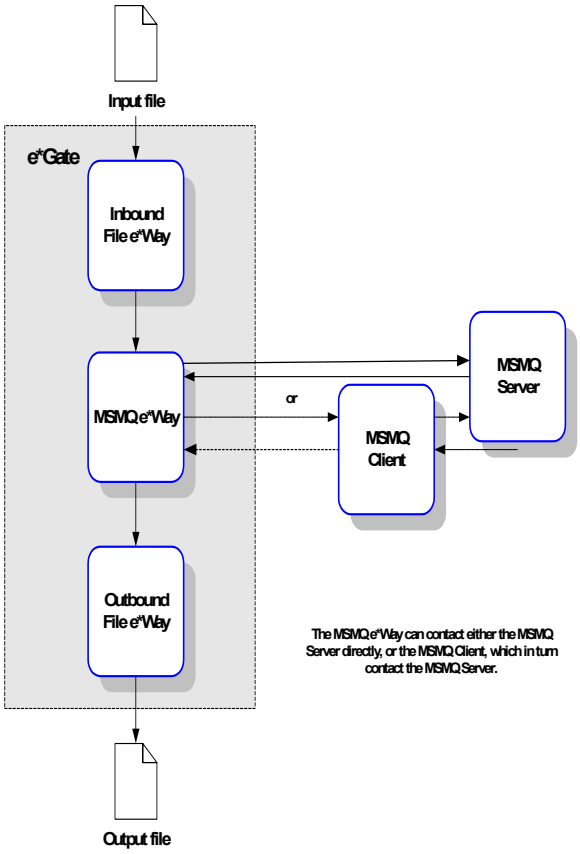
Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's online Help system.*

4.2 Creating the Sample Schema

In the sample schema, data is drawn from a text file using the inbound e*Way and sent to an external system using the MSMQ e*Way. The data returned from the external system is received by the MSMQ e*Way, then forwarded to another outbound e*Way and stored in an output file on the local system.

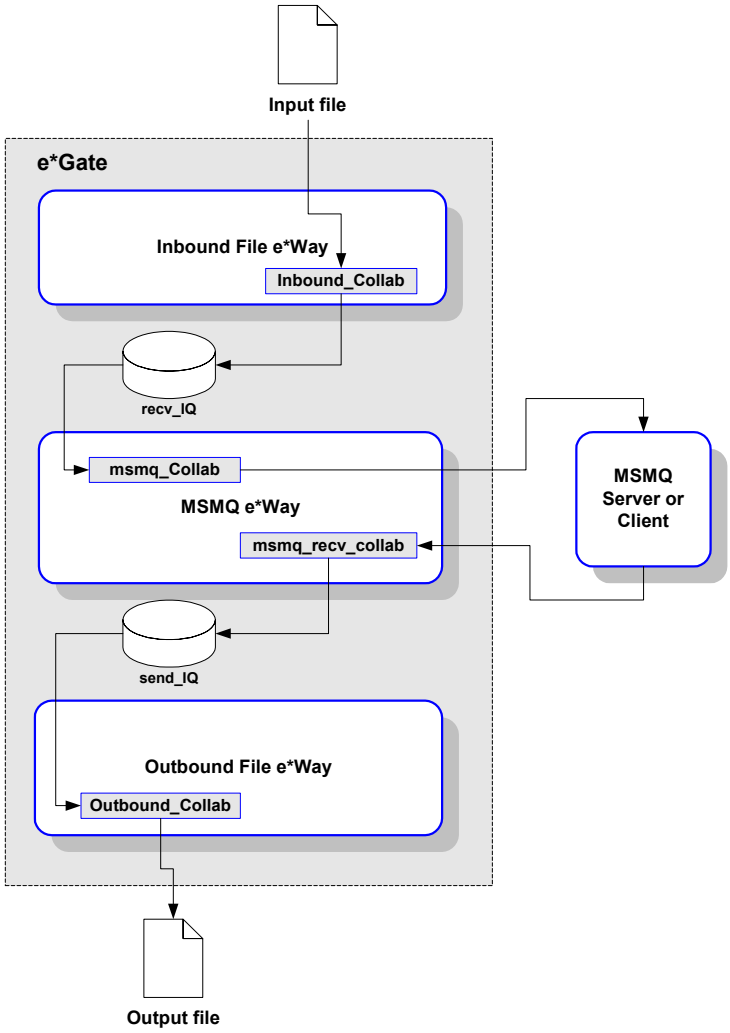
The following diagram illustrates the flow of data in the sample schema:

Figure 8 Sample Schema Basic Architecture



The schema requires a number of components as illustrated in the following diagram:

Figure 9 Components in the Sample Schema



4.2.1 Defining Event Types

1. Log into the e*Gate Schema Designer and create a new schema.

   The e*Gate Schema Designer's main screen appears.
2. In the Navigator, select the **Components** tab if it is not already selected.
3. In the Navigator, select the **Event Types** folder.
4. Create a new Event Type called **blob**.
5. Display the properties for Event Type **blob**.
6. Under **Event Type Definition** click **Find**.
7. Navigate to the **common** folder and assign the file **GenericIn.ssc**.
8. Click **OK** in the Properties dialog box to return to the e*Gate Schema Designer's main screen.

4.2.2 Defining Collaboration Rules

1. In the Navigator, select the **Collaboration Rules** folder.
2. Create a new Collaboration Rules component called **pass**.
3. Edit the properties of **pass** as follows:

Service	Pass through
Subscription	**blob** (the Event Type defined in step 4 above in **"Defining Event Types" on page 34**)
Publication	**blob** (the Event Type defined in step 4 above in **"Defining Event Types" on page 34**)

4.2.3 Defining IQs

1. In the Navigator, double-click the **Participating Hosts** folder, then double-click the host, then double-click the Control Broker and select the IQ Manager.
2. Display the properties for the IQ Manager and configure it to start automatically.
3. Create the following Standard SeeBeyond IQs:
   - recv_queue
   - send_queue
4. Display the properties for each IQ and verify that it uses the SeeBeyond Standard IQ Service.

4.2.4 Defining e*Ways and Collaborations

The next part of the procedure requires that you create and define the properties of a file-based e*Way.

Inbound e*Way

To create the Inbound e*Way:

1. In the Navigator, double-click the **Participating Hosts** folder, then double-click the host and select the Control Broker.
2. Create a new e*Way component called **Inbound**.
3. Display the e*Way's properties.
4. On the **General** tab, under **Executable file**, click **Find** to assign the file **stcewfile.exe**.
5. Select the **Start Up** tab. Configure the e*Way to start automatically.

To configure the e*Way's parameters:

1. Return to the **General** tab. Under **Configuration file**, click **New** to launch the e*Way Editor.
2. Configure the e*Way's parameters in the e*Way Editor as follows (parameters not listed in the table should retain their default values):

Section	Parameter: Setting
General Settings	**AllowIncoming**: Yes **AllowOutgoing**: No
Poller(inbound)Settings	**Polldirectory**:C:\INDATA\MSMQ (or other suitable directory) InputFileMask: leave unchanged

3. Save the settings and promote the file to run time.
4. In the e*Way's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

To create the Collaboration associated with the Inbound e*Way:

1. In the Navigator, select the e*Way **Inbound**.
2. Create a Collaboration called **inbound_collab**.
3. Display the Collaboration's properties and edit them as follows:

Collaboration Rules	***pass*** (pass through)
Subscription	Event Type: ***blob*** Source: ***<External>***
Publication	Event Type: ***blob*** Destination: ***send_queue***

4. In the Collaboration's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

MSMQ e*Way

To create the MSMQ e*Way:

1. In the Navigator, double-click the **Participating Hosts** folder, then double-click the host and select the Control Broker.
2. Create a new e*Way component called **MSMQ**.
3. Display the e*Way's properties.
4. On the **General** tab, under **Executable file**, click **Find** to assign the file **stcewgenericmonk.exe**.
5. Select the **Start Up** tab. Configure the e*Way to start automatically.

To configure the e*Way's parameters:

1. Return to the **General** tab. Under **Configuration file**, click **New** to launch the e*Way Editor.
2. Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewmsmq**.
3. Configure the e*Way's parameters in the e*Way Editor as shown in Table 2; parameters not listed in the table retain their default values.

Table 2 MSMQ e*Way Configuration Parameters

Section	Parameter: Setting
MSMQ Settings	**GetQueue**: Set to the Global Unique Identifier (GUID) for the queue to get information. (Gettranstype 0 or 3) **PutQueue**: Follow the format: myservername\\myqueuename (Puttranstype 0 or 3)

4. Save the settings and promote the file to run time.
5. In the e*Way's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

To create the first MSMQ Collaboration:

1. In the Navigator, select the e*Way **MSMQ**.
2. Create a Collaboration called **msmq_collab**.
3. Display the Collaboration's properties and edit them as follows:

Collaboration Rules	***pass*** (pass through)
Subscription	Event Type: ***blob*** Source: ***inbound_collab***
Publication	Event Type: ***blob*** Destination: ***<External>***

4. In the Collaboration's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

To create the next MSMQ Collaboration:

1. In the Navigator, select the e*Way **MSMQ**.
2. Create a Collaboration called **msmq_recv_collab**.
3. Display the Collaboration's properties and edit them as follows:

Collaboration Rules	***pass*** (pass through)
Subscription	Event Type: ***blob*** Source: ***<External>***

Publication	Event Type: ***blob*** Destination: ***recv_queue***

4 In the Collaboration's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

Outbound e*Way

To create the Outbound e*Way:

1 In the Navigator, double-click the **Participating Hosts** folder, then double-click the host and select the Control Broker.

2 Create a new e*Way component called **Outbound**.

3 Display the e*Way's properties.

4 On the **General** tab, under **Executable file**, click **Find** to assign the file **stcewfile.exe**.

5 Select the **Start Up** tab. Configure the e*Way to start automatically.

To configure the e*Way's parameters:

1 Return to the **General** tab. Under **Configuration file**, click **New** to launch the e*Way Editor.

2 Configure the e*Way's parameters in the e*Way Editor as follows (parameters not listed in the table should retain their default values):

Section	Parameter: Setting
General Settings	**AllowIncoming**: No AllowOutgoing: Yes
Outbound (sender) Settings	**Output directory**: C:\Data\MSMQ (or some other suitable directory) **Output File Name**: msmq_out.txt

3 Save the settings and promote the file to run time.

4 In the e*Way's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

To create the Collaboration associated with the Outbound e*Way:

1 In the Navigator, select the e*Way **Outbound**.

2 Create a Collaboration called **outbound_collab**.

3 Display the Collaboration's properties and edit them as follows:

Collaboration Rules	***pass*** (pass through)
Subscription	Event Type: ***blob*** Source: ***msmq_recv_collab***
Publication	Event Type: ***blob*** Destination: ***<External>***

4 In the Collaboration's Properties dialog box, click **OK** to save all changes and return to the e*Gate Schema Designer's main window.

This completes schema configuration in the e*Gate Schema Designer.

4.2.5 Running the Schema

To run the schema, ensure that the Control Broker is running and create a sample input text file. Unless you specified something different in the e*Way Editor, make sure you save the file in the **C:\INDATA\MSMQ** directory with a **.dat** extension. After you create the file and save it to the directory, e*Gate will return an output file (**msmq_outputn.dat**, where *n* is a number) in the **C:\DATA\MSMQ** directory (or whatever directory you specified).

Creating Sample Data for the Schema

To pass data to the MSMQ e*Way, generate a data file as below or any other text file. The script below shows an example.

```
0,'String length15','STR',0,'F',0,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
1,'String length15','STR',1,'F',1,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
2,'String length15','STR',2,'F',2,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
3,'String length15','STR',3,'F',3,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
4,'String length15','STR',4,'F',4,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
5,'String length15','STR',5,'F',5,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
6,'String length15','STR',6,'F',6,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
7,'String length15','STR',7,'F',7,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
8,'String length15','STR',8,'F',8,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
9,'String length15','STR',9,'F',9,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
10,'String length15','STR',10,'F',10,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
11,'String length15','STR',11,'F',11,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
12,'String length15','STR',12,'F',12,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
13,'String length15','STR',13,'F',13,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
14,'String length15','STR',14,'F',14,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
15,'String length15','STR',15,'F',15,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
16,'String length15','STR',16,'F',16,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
17,'String length15','STR',17,'F',17,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
18,'String length15','STR',18,'F',18,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
19,'String length15','STR',19,'F',19,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
20,'String length15','STR',20,'F',20,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
21,'String length15','STR',21,'F',21,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
22,'String length15','STR',22,'F',22,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
23,'String length15','STR',23,'F',23,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
24,'String length15','STR',24,'F',24,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
25,'String length15','STR',25,'F',25,'String length15','LEN1','STRING','STRING','Length 10 ','LEN1'
```

4.3 Sample Monk Scripts

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

The syntax of the **stctrans** utility is

```
stctrans monk_file.monk
```

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

4.3.1 *Monk Script Example (mqsamp.monk)*

The script below contains the "Direct=OS:titanium\\test" format as part of the hard-coded script. The script contains both an example of a **Get** operation as well as a **Put** operation.

```
(load-extension "monk_msmq.dll")
;; How to open a QUEUE
;;
(define puth (MSMQOPEN "DIRECT=OS:titanium\\test" 2 0))
(display puth)

;; How to put to a QUEUE (use handle from MSMQOPEN)
;;
;; 1st arg is the Queue Handle
;; 2nd arg is the message
;; optional 3rd arg is the transaction level (0, or 3 - see
;; below)
;; optional 4th arg is the label used for the put (text)
;;
(display (MSMQPUT puth "12345678"))(newline)


;; How to get from QUEUE (use handle from MSMQOPEN)
;;

(define geth (MSMQOPEN "PUBLIC=274059E7-6A46-11D3-B5A0-0004AC9035BE" 1 0))
(display geth)


;; How to get from a QUEUE (use handle from MSMQOPEN)
;;
;; 1st arg is the Queue Handle
;; 2nd arg is the number of msec to wait for a message
;; optional 3rd arg is the transaction level or type:
;;           0 = no transaction support
;;           3 = part of single message transaction
;;
;; If fails, returns #f
;; If no data (timeout occurred on wait) return #t
;; If data available, returns object of type string

(define msg (MSMQGET geth 1000 0))
(if (string? msg)
  (begin
    (display "Received message:\n")
    (display msg)
    (newline)
  )
  (begin
    (if (eq? msg #t)
      (begin
        (display "NO DATA\n")
      )
      (begin
        (display "GET failed\n")
      )
    )
  )
)

(display (MSMQCLOSE geth))
(display (MSMQCLOSE puth))
```

4.3.2 *Monk Script Example (msmqread.monk)*

The script below contains the "PUBLIC=274059E7-6A46-11D3-B5A0-0004AC9035BE" format as part of the hard-coded script. The script perfoms a **Get** operation.

```
(load-extension "monk_msmq.dll")
(display "Running...\n")

(define x (MSMQOPEN "PUBLIC=274059E7-6A46-11D3-B5A0-0004AC9035BE" 1 0))

(do ((i 0 (+ i 1))) ((= i -1))
 (define r (MSMQGET x 1000))
 (display i)(display ">")
 (if (string? r)
  (display (string-length r))
 )
 (newline)
)

(display (MSMQCLOSE x))(newline)
```

4.3.3 *Monk Script Example (msmqwrit.monk)*

The script below contains the "DIRECT=OS:MSMQServer\\queuename" format as part of the hard-coded script. The script perfoms a **Put** operation.

```
(load-extension "monk_msmq.dll")
(display "Running...\n")

;;  MSMQPUT syntax:
;; 1st arg is the Queue Handle
;; 2nd arg is the bytes to put to the queue
;; optional 3rd arg is the transaction level or type:
;;           0 = no transaction support
;;           3 = part of single message transaction
;; optional 4th arg is the text label used during the put
;;

(define x (MSMQOPEN "DIRECT=OS:titanium\\test" 2 0))
(display "1")(display x)(newline)
(do (( i 0 (+ i 1))) ((> i 1000000))
  (display (MSMQPUT x (string-append (format "%08d" (number->string i))
"[12345678901234567890123456789012345678901234567890]") 0 "STC"))(newline)
)

(display (MSMQCLOSE x))(newline)
```

Chapter 5

MSMQ e*Way Functions

This chapter describes the functions used to control the MSMQ e*Way's basic operations.

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

5.1 Basic Functions

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

The basic functions are

start-schedule

Syntax

```
(start-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

stop-schedule requests that the e*Way halt execution of the "Exchange Data with External" function specified within the e*Way's configuration file. Execution will be stopped when the e*Way concludes any open transaction. Does not 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 completes the e*Gate shutdown procedure that was initiated by the Control Broker but was interrupted by returning a non-null value within the Shutdown Command Notification Function (see **"Shutdown Command Notification Function" on page 28**). Once this function is called, 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 Standard e*Way Functions

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

The current suite of Standard e*Way functions included is:

msmq-conn-establish

Syntax

```
(msmq-conn-estab)
```

Description

msmq-conn-estab establishes a connection to the external system.

Parameters

None.

Return Values

string

"UP"or "SUCCESS" indicates the connection is established. Anything else indicates no connection.

Throws

None.

Additional Information

See **"External Connection Establishment Function" on page 26** for more information.

msmq-conn-shutdown

Syntax

```
(msmq-conn-shutdown suspend_notification)
```

Description

msmq-conn-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 45) call from within a Monk function to allow the requested shutdown to process to continue.

Parameters

Name	Type	Description
suspend_notification	string	When the e*Way calls this function, it will pass the string "SUSPEND_NOTIFICATION" as the parameter.

Return Values

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

Throws

None.

Additional Information

See **"External Connection Shutdown Function" on page 27** for more information.

msmq-conn-verify

Syntax

```
(msmq-conn-verify)
```

Description

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

Parameters

None.

Return Values

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

Throws

None.

Additional Information

See **"External Connection Verification Function" on page 26** for more information.

msmq-incoming

Syntax

```
(msmq-incoming)
```

Description

msmq-incoming 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 25** for more information.

Examples

```
(define msmq-incoming
 (lambda (  )
  (let ((result ""))
  (display "[++] Executing e*Way external data exchange function.")
 result
  )
))
```

msmq-init

Syntax

```
(msmq-init)
```

Description

msmq-init begins the initialization process for the e*Way. This function loads the **stc_monkmsmq.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 23** for more information.

msmq-neg-ack

Syntax

```
(msmq-neg-ack message-string)
```

Description

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

Parameters

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

Return Values

string
An empty string indicates a successful operation.

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

Throws

None.

Additional Information

See **"Negative Acknowledgment Function" on page 28** for more information.

Examples

```
(define msmq-neg-ack
 (lambda ( message-string )
  (let ((result ""))
 ( (display "[++] Executing e*Way external negative acknowledgment
function.")
 (display message-string)
 result
  )
))
```

msmq-outgoing

Syntax

```
(msmq-outgoing event-string)
```

Description

msmq-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 message to be immediately resent. The e*Way will compare the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
- If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.

"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 44 for more information on this function.

If a string other than the preceding 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.

Throws

None.

Additional Information

See **"Process Outgoing Message Function" on page 24** for more information.

Examples

```
(define msmq-outgoing
 (lambda ( message-string )
  (let ((result ""))
  (display "[++] Executing e*Way external process outgoing message
function.")
 (display message-string)
 result
  )
))
```

msmq-pos-ack

Syntax

```
(msmq-pos-ack message-string)
```

Description

msmq-pos-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	Description
message-string	The Event for which an acknowledgment is sent.

Return Values

A 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.

Throws

None.

Additional Information

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

Examples

```
(define msmq-pos-ack
 (lambda ( message-string )
  (let ((result ""))
  (display "[++] Executing e*Way external positive acknowledgment
function.")
 (display message-string)
 result
  )
))
```

msmq-shutdown

Syntax

```
(msmq-shutdown shutdown_notification)
```

Description

msmq-shutdown notifies the external system that the e*Way is shutting down.

Parameters

Name	Type	Description
shutdown-notification	string	When the e*Way calls this function, it will pass the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

string

SUCCESS allows an immediate shutdown to occur, anything else delays shutdown until a **shutdown-request** is executed successfully**.**

Throws

None.

Additional Information

See **"Shutdown Command Notification Function" on page 28** for more information.

Examples

```
(define msmq-shutdown
 (lambda ( message-string )
  (let ((result "SUCCESS"))
  (display "[++] Executing e*Way external shutdown command
notification function.")
 result
  )
```

```
))
```

msmq-startup

Syntax

```
(msmq-startup)
```

Description

msmq-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 24** for more information.

Examples

```
(define msmq-startup
 (lambda (  )
  (let ((result "SUCCESS"))
  (display "[++] Executing e*Way external startup function.")
 result
  )
))
```

5.3 MSMQ e*Way Functions

The current suite of native MSMQ e*Way functions is:

MSMQBEGINTRANS

Syntax

```
(MSMQBEGINTRANS queuehandle)
```

Description

MSMQBEGINTRANS initiates an internal MSMQ transaction.

Parameters

Name	Type	Description
queuehandle	opaque handle	The MSMQ handle returned by MSMQOPEN.

Return Values

Boolean

Returns **#t** (true) if the handle is valid; otherwise, returns **#f** (false).

Throws

None.

Additional Information

The following considerations must be made:

- **MSMQOPEN** must be called before **MSMQBEGINTRANS**.
- **MSMQBEGINTRANS** must be used when putting to or getting from an internal transaction queue.
- **MSMQCOMMIT** must be used in conjunction with **MSMQBEGINTRANS**.
- When using **MSMQBEGINTRANS**, the **dwTransType** parameter in the **MSMQGET** and **MSMQPUT** functions must be passed a value, but that value is over-ridden by **MSMQBEGINTRANS**.

MSMQCLOSE

Syntax

```
(MSMQCLOSE hMSMQ)
```

Description

MSMQCLOSE closes the connection with an MSMQ queue.

Parameters

Name	Type	Description
hMSMQ	opaque handle	The MSMQ handle returned by MSMQOPEN.

Return Values

Boolean

Returns **#t** (true) if the queue closed successfully; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(display (MSMQCLOSE geth))
```

or

```
(display (MSMQCLOSE puth))
```

MSMQCOMMIT

Syntax

```
(MSMQCOMMIT queuehandle)
```

Description

MSMQCOMMIT commits a transaction.

Parameters

Name	Type	Description
queuehandle	opaque handle	The MSMQ handle returned by MSMQOPEN.

Return Values

Boolean

Returns **#t** (true) if the handle is valid; otherwise, returns **#f** (false).

Throws

None.

Additional Information

MSMQCOMMIT must follow the call for **MSMQOPEN**, **MSMWBEGINTRANS**, and either an **MSMQGET** or **MSMQPUT**.

MSMQGET

Syntax

```
(MSMQGET hMSMQ cmsTimeout [dwTransactionType])
```

Description

MSMQGET gets a message from an MS queue.

Parameters

Name	Type	Description
hMSMQ	opaque handle	The MSMQ handle returned by MSMQOPEN, (must have iOPTION parameter set to 1 - receive).
cmsTimeout	integer	Specifies the amount of time in milliseconds between attempts to retrieve messages.
dwTransactionType	integer	An optional parameter specifying the transaction type; 0 = No transaction support, 3 = part of single message transaction.

Return Values

Boolean

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

Throws

None.

Additional Information

The following variables are created and defined as Monk variables after each call:

- MSMQ_ATTR_MSGLABEL
- MSMQ_ATTR_MSGID

Monk Variable

Name	Type	Description
MSMQ_ATTR_MSGLABEL	string (maximum length is 250 Unicode characters including the end-of-string character)	Specifies the label for the message.
MSMQ_ATTR_MSGID	20 byte binary array	Specifies the message identifier which is composed of the machine GUID of the computer that sent the message, and a unique identifier for the message.

If the value of the MSMQ_ATTR_MSGLABEL is not defined or is empty, the message does not contain a label.

Examples

```
((define x (MSMQOPEN "PUBLIC=274059E7-6A46-11D3-B5A0-0004AC9035BE" 1 0))

(do ((i 0 (+ i 1))) ((= i -1))
 (define r (MSMQGET x 1000))
 (display i)(display ">")
 (if (string? r)
  (display (string-length r))
 )
 (newline)
)
)
(display (MSMQCLOSE x))(newline)
```

MSMQOPEN

Syntax

(MSMQOPEN *pszQueueReference iOption iSharing*)

Description

MSMQOPEN opens an MS queue and returns the MSMQ handle.

Parameters

Name	Type	Description
pszQueueReference	string	A valid MSMQ queue name.
iOption	integer	Specifies the transaction value selection: 1 = receive, 2 = send.
iSharing	integer	Specifies share selection, 0 = None ; 1 = deny record share.

Return Values

handle

Returns the MSMQ opaque handle to be accessed by the other functions.

Throws

None.

Examples

```
(define puth (MSMQOPEN "DIRECT=OS:titanium\\test" 2 0))
(display puth)


(define geth (MSMQOPEN "PUBLIC=274059E7-6A46-11D3-B5A0-0004AC9035BE"
1 0))
(display geth)
```

In the first example, both the pszQueueReferenceName and the iOption show that this handle will be used with MSMQPUT.

In the same fashion, in the second example, the same parameters indicate that this handle will be used with MSMQGET.

MSMQPUT

Syntax

```
(MSMQPUT hMSMQ pszData dwTransactionType pszLabel)
```

Description

MSMQPUT puts a message into an MS queue.

Parameters

Name	Type	Description
hMSMQ	opaque handle	The MSMQ handle returned by MSMQOPEN, (must have iOPTION parameter set to 2 - send).
pszData	string	The data to be sent.
dwTransactionType	integer	An optional parameter specifying the transaction type; 0 = No transaction support, 3 = part of single message transaction.
pszLabel	string	An optional parameter, sets property ID label on message.

Return Values

Boolean

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

Throws

None.

Examples

```
((define x (MSMQOPEN "DIRECT=OS:titanium\\test" 2 0))
(display "1")(display x)(newline)
(do (( i 0 (+ i 1))) ((> i 1000000))
  (display (MSMQPUT x (string-append (format "%08d" (number->string i))
"[12345678901234567890123456789012345678901234567890]") 0 "STC"))(newline)
)

(display (MSMQCLOSE x))(newline)
)
```

MSMQ-QUEUEHANDLE?

Syntax

```
(MSMQ-QUEUEHANDLE? hMSMQ)
```

Description

MSMQ-QUEUEHANDLE verifies that the handle name specified is a valid MSMQ handle.

Parameters

Name	Type	Description
hMSMQ	opaque handle	The MSMQ handle returned by MSMQOPEN.

Return Values

Boolean
Returns **#t** (true) if the handle is valid; otherwise, returns **#f** (false).

Throws

None.

MSMQROLLBACK

Syntax

```
(MSMQROLLBACK queuehandle)
```

Description

MSMQROLLBACK performs a rollback on a transaction.

Parameters

Name	Type	Description
queuehandle	opaque handle	The MSMQ handle returned by MSMQOPEN.

Return Values

Boolean
Returns **#t** (true) if the handle is valid; otherwise, returns **#f** (false).

Throws

None.

Index

J

M

N

O

P

R

S

T

U

Z