SeeBeyond ICAN Suite

e*Way Intelligent Adapter for MQSeries User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Monk Version

Version 20051014142937.

Contents

Chapter 1

Introduction 6

Overview 6
- Intelligent Queues and IQ Managers 6
- Intended Reader 7
- Components 7

Supported Operating Systems 7

System Requirements 7

External System Requirements 8

Chapter 2

Installation 9

Windows Installation 9
- Pre-installation 9
- Installation Procedure 9

UNIX Installation 10
- Pre-installation 10
- Installation Procedure 10
- AIX Systems 11

Files/Directories Created by the Installation 11

Chapter 3

Configuration 13

e*Way Configuration Parameters 13
- General Settings 13
  - Journal File Name 14
  - Max Resends Per Message 14
  - Max Failed Messages 14
  - Forward External Errors 14
- Communication Setup 15
  - Start Exchange Data Schedule 15
  - Stop Exchange Data Schedule 15
  - Exchange Data Interval 16

Down Timeout 16
Up Timeout 16
Resend Timeout 17
Zero Wait Between Successful Exchanges 17
Monk Configuration 17
Operational Details 19
How to Specify Function Names or File Names 25
Additional Path 26
Auxiliary Library Directories 26
Monk Environment Initialization File 27
Startup Function 27
Process Outgoing Message Function 28
Exchange Data with External Function 28
External Connection Establishment Function 29
External Connection Verification Function 30
External Connection Shutdown Function 30
Positive Acknowledgment Function 31
Negative Acknowledgment Function 31
Shutdown Command Notification Function 32
MQ Settings 32
Local MQ Install 33
Qmanager 33
Queue Name 33
MQ Get Buffer Length 33
Enable Connection Test 34
Connection Test Queue Name 34
Environment Variable 34
Environment Configuration 34
External Configuration Requirements 35

Chapter 4
Implementation 36
Implementation Notes 36
Connecting to a Remote Queue 37
Error Handling 37
Installing the MQSeries (Monk) Sample Schema 37
Install the Sample Schema on the Registry Host 38

Chapter 5
MQSeries e*Way Functions 39
Basic Functions 39
MQSeries e*Way External Init Functions 43
MQSeries Monk Functions 50

MQSeries Auxiliary Functions 58
MQSeries Structures 68
MQSeries Structure Related Functions 72

Index 74

Chapter 1

Introduction

This chapter introduces you to SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way Intelligent Adapter for the MQSeries. It includes an overview of this manual, and a list of system requirements for installation.

1.1 Overview

The MQSeries e*Way (Monk version) enables the e*Gate system to exchange data with IBM's MQSeries versions 5.0, 5.1, 5.2 and 5.3 (WebSphere MQ V5.3). IBM MQSeries is "middleware" that provides commercial messaging and queuing services. Messaging enables programs to communicate with each other via messages instead of direct connection. Placing these messages in queues for temporary storage frees up programs to continue to work independently. This process also allows communication across a network of unlike components, processors, operating systems, and protocols.

1.1.1. Intelligent Queues and IQ Managers

A key component within e*Gate is the Intelligent Queue, which provides storage for data while inside the e*Gate system. Data from Event Types are inserted and retrieved when the Control Broker invokes an e*Way. The IQ Manager controls the different IQs. The MQSeries e*Way allows e*Gate to seamlessly exchange data with applications that are MQSeries enabled.

The MQSeries IQ allows e*Gate to leverage existing MQSeries operational and management infrastructure for its underlying persistent storage. The MQSeries IQ makes use of queuing capability, but does not make use of MQSeries channels.

Queuing allows applications to run independently of one another, at different speeds and times. Applications can send messages to a queue and get messages from a queue at any time.

The MQSeries e*Way is configurable and transparently integrates existing systems and databases to the MQSeries, through e*Gate. This document explains how to install and configure the MQSeries e*Way.

1.1.2. Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have a high level of experience with Windows or UNIX operations and administration; to be thoroughly familiar with IBM's MQSeries and with Windows-style GUI operations.

1.1.3. Components

The MQSeries e*Way comprises the following:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk functions (discussed in **Chapter 5**)
- Dynamic Link Library (DLL) files

A complete list of installed files appears in **Table 1 on page 12**.

1.2 Supported Operating Systems

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

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0 and 11.i (PA-RISC)
- IBM AIX 5L version 5.1, 5.2, and 5.3
- Sun Solaris 8 and 9
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0 and 11i (PA-RISC)
- Japanese IBM AIX 5L version 5.1, 5.2, and 5.3
- Japanese Sun Solaris 8 and 9

Note: *For AIX, see* **AIX Systems** *on page 11* *for information on required patches.*

1.3 System Requirements

To use the MQSeries 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.

1.4 External System Requirements

To use the MQSeries Monk e*Way, you need the following:

- IBM MQSeries version 5.0, 5.1, 5.2 or 5.3 (WebSphere MQ V5.3).
- For **AIX** systems, MQSeries requires a patch, available from IBM at:

  ftp://ftp.software.ibm.com/software/mqseries/fixes

  The README.TXT file contains information to help you determine which patches are applicable to your specific system. See **AIX Systems** on page 11 for more information.

Chapter 2

Installation

This chapter explains how to install the MQSeries 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 MQSeries e*Way on a Windows system

1. Log in as an Administrator to 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 launches. 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 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 **e*Gate Integrator User's Guide**.*

2.2 UNIX Installation

2.2.1. Pre-installation

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

2.2.2. Installation Procedure

To install the MQSeries e*Way on a UNIX system

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

2 If necessary, mount the CD-ROM drive.

3 At the shell prompt, type:

cd /cdrom/setup

4 Start the installation script by typing:

setup.sh

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

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

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

11 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 **e*Gate Integrator User's Guide**.*

2.2.3. AIX Systems

For AIX systems, MQSeries requires a patch, available from IBM at:

ftp://ftp.software.ibm.com/software/mqseries/fixes

Open the folder that applies to your specific AIX version and see the README file for information that will help you determine which patches apply to your specific system.

For AIX 5L version 5.1

From the above web site, locate the aix51/U471218 folder. The README.TXT file contains information that will help you determine which of the two patches included in this folder are applicable to your specific system.

We recommend that AIX 5.1 users install the **U471218.tar.Z** patch, if it has not already been applied.

2.3 Files/Directories Created by the Installation

The MQSeries e*Way installation process will install the files shown in Table 1 below within the e*Gate **client** 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

Install Directory	Files
bin\	stcewgenericmonk.exe stc_monkmqclient.dll stc_monkmqserver.dll
configs\stcewgenericmonk\	stcewmq.def
monk_library\MQ\	MQclient.monk MQserver.monk mq-stdver-eway-funcs.monk

Select the appropriate **.dll** file based on whether the installed MQSeries e*Way will be in server or client mode. The same applies to selecting the appropriate ***.monk** file to use. All other files displayed apply to both client and server modes.

Caution: *The **monk_MQclient.dll** and **monk_MQserver.dll** files cannot be run simultaneously by the same executable file. Trying to do so will cause run-time conflicts.*

Chapter 3

Configuration

This chapter provides instructions for configuring the MQSeries e*Way. Information on configuration parameters for standard file-based e*Ways is available in the **Standard e*Way Intelligent Adapter User's Guide**.

3.1 e*Way Configuration Parameters

The component e*Way configuration parameters are set, using the e*Way Editor. The MQSeries e*Way requires two component e*Ways to communicate to and from MQSeries. The e*Ways

To edit component 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 *e*Gate Integrator User's Guide*.

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

- General Settings
- Communication Setup
- Monk Configuration
- MQ Settings

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 file name, 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 (package of data) 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 14** for more information.)

Max Resends Per Message

Description

Specifies the number of times the e*Way will attempt to resend an 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 Events (messages) that the e*Way will allow. When the specified number of failed messages is reached, the e*Way will shut down and exit.

Required Values

An integer between 1 and 1,024. The default is 3.

Forward External Errors

Description

Selects whether error messages that begin with the string "DATAERR" that are received from the external system will be queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 28** 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.*

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 functions:

- **Exchange Data With External**
- **Positive Acknowledgment**
- **Negative Acknowledgment**

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 28**, **"Exchange Data Interval" on page 16**, and **"Stop Exchange Data Schedule" on page 15** 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 16** and **"Stop Exchange Data Schedule" on page 15** 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 29** 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 30** 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 an Event (message) 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**.

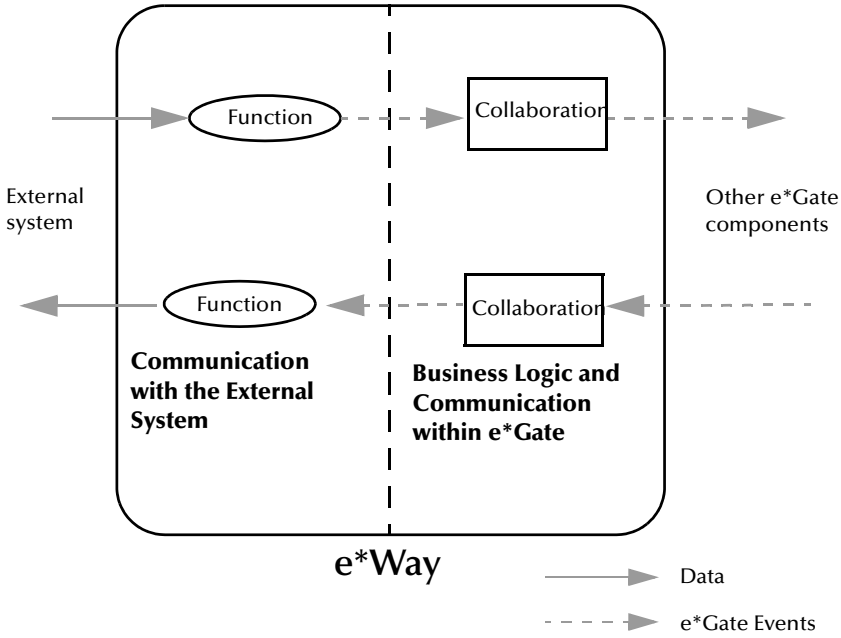
See **"Exchange Data with External Function" on page 28** for more information.

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 on page 18**) 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 side 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 may create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as **Notepad** or UNIX **vi**).

The communications side 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.

Note: *Two separate e*Ways are required when reading from and writing to MQSeries.* ***Sending*** *e*Ways must be set as* ***client****, and* ***receiving*** *e*Ways must be set as* ***server****, in accordance with the IBM standard for MQSeries and to use the appropriate DLLs. The nature of the server is to receive, waiting for incoming connections and data, whereas the nature of the client is to solicit data and send it out. IBM has created separate libraries (DLLs) for sending and receiving. Function sets are similar but the underlying behaviors are inherently different.*

Operational Details

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

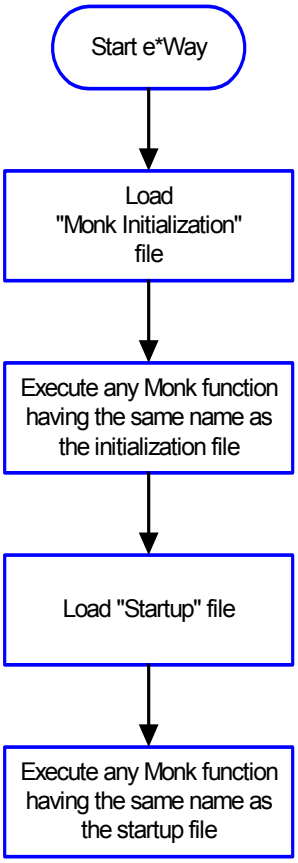
Type of Operation	Name
Initialization	**Startup Function** on page 27 (also see **Monk Environment Initialization File** on page 27)
Connection	**External Connection Establishment Function** on page 29 **External Connection Verification Function** on page 30 **External Connection Shutdown Function** on page 30
Schedule-driven data exchange	**Exchange Data with External Function** on page 28 **Positive Acknowledgment Function** on page 31 **Negative Acknowledgment Function** on page 31
Shutdown	**Shutdown Command Notification Function** on page 32
Event-driven data exchange	**Process Outgoing Message Function** on page 28

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

Initialization Functions

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

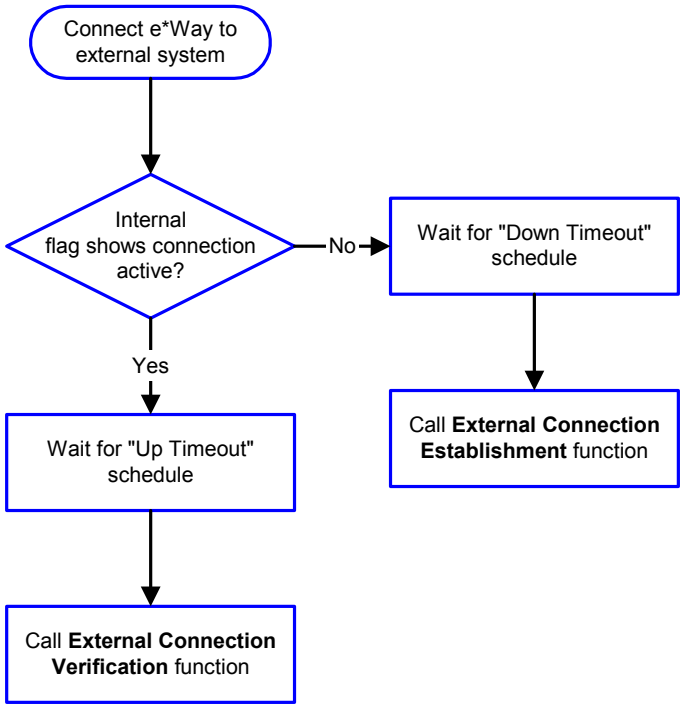
Figure 2 Initialization Functions



Connection Functions

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

Figure 3 Connection Establishment and Verification Functions

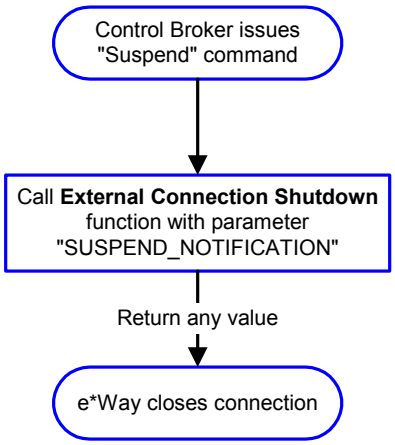


Note: *The e*Way selects the connection function based on an internal* ***up/down*** *flag rather than a poll to the external system. See* **Figure 5 on page 23** *and* **Figure 7 on page 25** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** ***on page 41*** *and* **send-external-down** ***on page 41*** *for more information.*

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

Figure 4 Connection Shutdown Function



Schedule-driven Data Exchange Functions

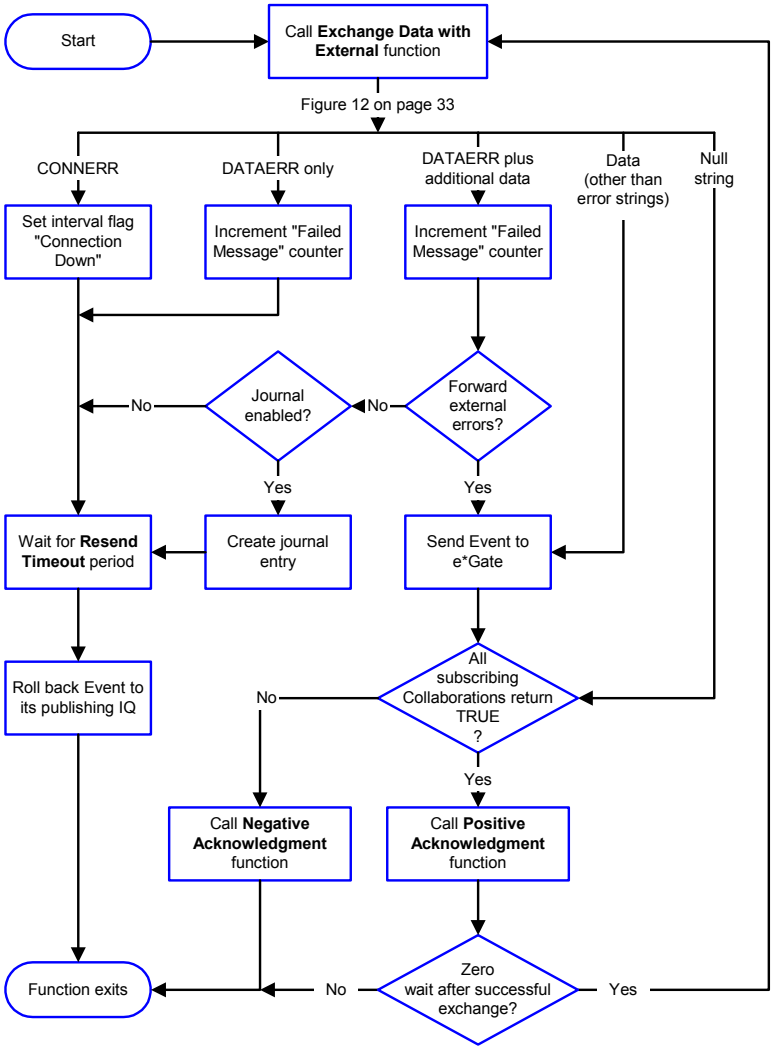
Figure 5 on page 23 illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

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

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

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

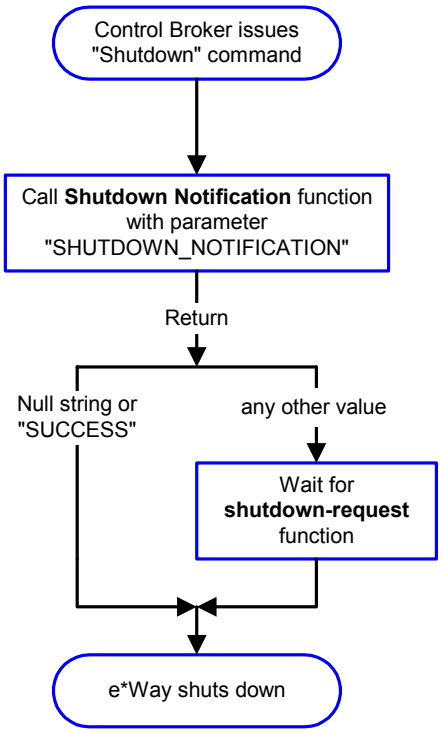
Figure 5 Schedule-driven Data Exchange Functions



Shutdown Functions

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

Figure 6 Shutdown Functions



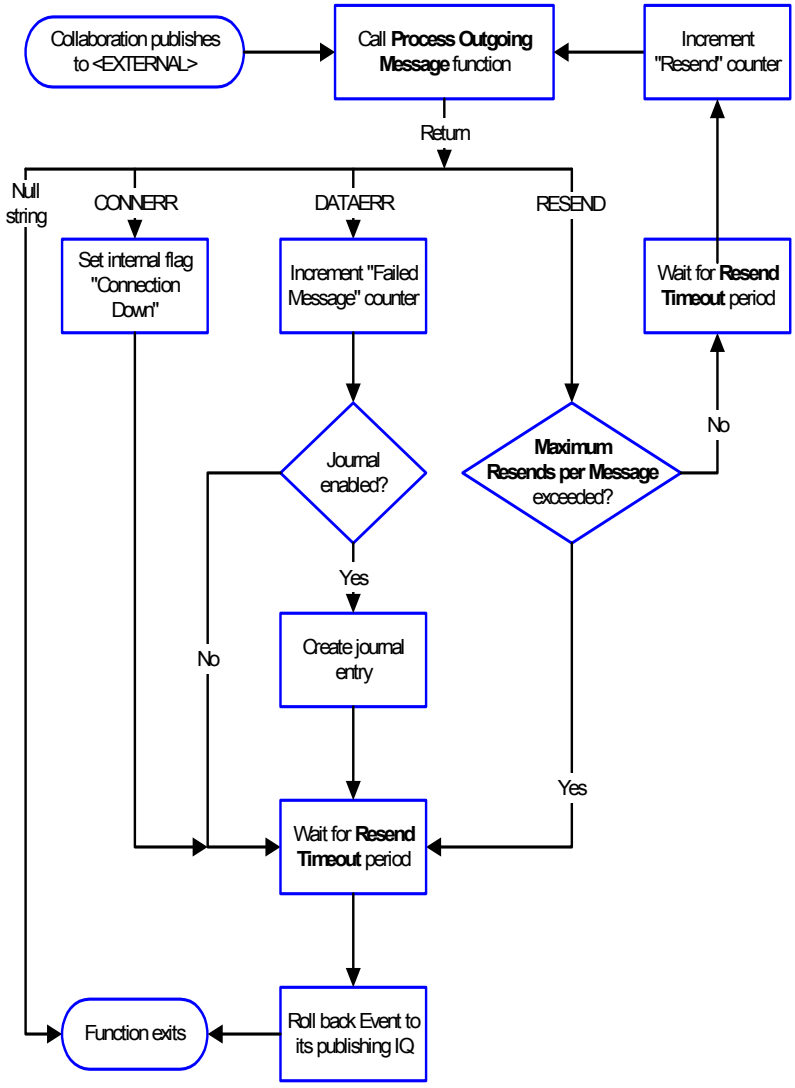
Event-driven Data Exchange Functions

Figure 7 on page 25 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 path name, or a series of paths separated by semicolons. This parameter is optional and may be left blank.

Additional Information

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

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

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

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

Auxiliary Library Directories

Description

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

Required Values

A path name, or a series of paths separated by semicolons.

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/MQ**.

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 file name within the load path, or file name 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 26** for more information about the load path.

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 default is **MQ-stdver-init**.

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 2 on page 20**).

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.

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 20**). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

The default is **MQ-stdver-startup**.

Process Outgoing Message Function

Description

Specifies the Monk function responsible for sending outgoing Events (messages) 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.*

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 25** 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.*

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.

Additional Information

The function accepts no input and must return a string (see **Figure 5 on page 23** 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 23** 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.

The default is **MQ-stdver-data-exchg**.

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

If this parameter is set to **Yes**, the e*Way 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.*

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 default is **MQ-stdver-conn-estab**.

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.

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 default is **MQ-stdver-conn-ver**.

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.

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.

The default is **MQ-stdver-conn-shutdown**.

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.

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).

The default is **MQ-stdver-pos-ack**.

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.

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).

The default is **MQ-stdver-neg-ack**.

Shutdown Command Notification Function

Description

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

Required Values

The name of a Monk function.

Additional Information

When the Control Broker issues a shutdown command to the e*Way, 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 41).

The default is **MQ-stdver-shutdown**.

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. MQ Settings

The MQ Settings section enables you to configure the parameters that support queuing and the connection to the IBM MQSeries. A message queue is a named destination to which messages can be sent. A message is a string of bytes that has meaning to the applications that use the message. MQSeries messages consist of two parts, a message descriptor and application data. The content and structure of the application data are defined by the application programs that use them. The message descriptor identifies the message and contains other control information or attributes, such as data, the time the message was created, type of message, and the priority assigned to the message.

Messages accumulate on queues until they are retrieved by programs that service those queues. A queue can be either a buffer area in the memory of a computer or a data set on a permanent storage device such as a disk.

Before an application can send messages, a queue manager and some queues must be created. Queues reside in, and are managed by, a queue manager. Queue managers have the responsibility for monitoring that

- Attributes are changed according to commands received.
- Events, such as trigger events are generated when the appropriate conditions are met.
- Messages are put into the correct queue, as requested by the application, or routed to another queue manager. When this cannot be accomplished, the application is informed and an appropriate error message code is provided.

Applications access queues only through the external services of the queue manager. These applications can open a queue, put messages on it, get messages from it, and close the queue.

Local MQ Install

Description

Use this setting to configure the e*Way as either client or server. If the MQSeries server software was installed on the host running this e*Way, set this parameter to **server**. If the MQSeries client software was installed on the host running this e*Way, set this parameter to **client**.

Required Values

The type of installation, either **client** or **server**. The default is **server**.

Qmanager

Description

Specifies the name of the IBM MQSeries queue manager.

Required Values

A queue manager's name.

Queue Name

Description

Specifies the MQSeries message queue name.

Required Values

A queue name.

MQ Get Buffer Length

Description

Specifies the maximum size of the MQSeries receive buffer.

Required Values

An integer between l,024 and 4,194,304. The default is 8192.

Enable Connection Test

Description

Specifies whether or not the e*Way should attempt a connection to the Connection Test Queue at a regular interval to verify the connection is still alive.

Required Values

When set to **enable**, the e*Way will attempt the connection. If set to **disable**, no connection attempt will be made.

Connection Test Queue Name

Description

Specifies the name of the MQSeries Connection Test Queue.

Required Values

A name that identifies the Test Queue Connection.

3.2 Environment Variable

When connecting to IBM MQSeries in Client mode, the MQServer environment variable must be set as follows:

- For **UNIX** enter the following:

```
setenv MQServer CHANNEL#/TCP/'<hostname>' or '<hostname(port
number)>' or '<server IP address>'
```

  Where # is the channel number.

  For example:

```
setenv MQServer CHANNEL2/TCP/'mainserver(3843)'
```

- For **Windows** open the Command Prompt window and enter the following:

```
set MQServer=CHANNEL#/TCP/'<hostname>' or '<hostname(port
number)>' or '<server IP address>'
```

  For example:

```
set MQServer=CHANNEL3/TCP/'192.168.0.2'
```

3.3 Environment Configuration

To support the operation of this e*Way, no changes are necessary

- In the Participating Host's operating environment
- In the e*Gate system

Note: *Changes to Monk files can be made using the Collaboration Rules Editor (available from within the Schema Designer) or with a text editor. However, if you use a text editor to edit Monk files directly, you* ***must*** *commit these changed files to the e*Gate Registry or your changes will not be implemented.*

*For more information about committing files to the e*Gate Registry, see the Schema Designer's online Help system, or the* ***stcregutil*** *command-line utility in the e*Gate Integrator System Administration and Operations Guide.*

3.4 External Configuration Requirements

There are no configuration changes required in the external system. All necessary configuration changes can be made within e*Gate.

Chapter 4

Implementation

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

4.1 Implementation Notes

Follow these steps to use the MQSeries e*Way:

1. Install the software required in this order. Follow the procedures for installing e*Gate as outlined in the SeeBeyond ICAN Suite Installation Guide.
2. Install the MQSeries e*Way from the **Add-Ons** folder on the CD.
3. Check to ensure the MQ Manager is up and running. Refer to the *MQ Quick Start Guide* (supplied by IBM) for instructions.
4. Ensure that the MQ Administrator creates a queue for the appropriate MQ Manager.
5. Configure the MQSeries e*Ways, as explained in **Chapter 3**.

   Two separate e*Ways are required when reading from and writing to MQSeries. **Sending** e*Ways must be set as **client** and **receiving** e*Ways must be set as **server**, in accordance with the IBM standard for MQSeries and to use the appropriate DLLs. The nature of the server is to receive, waiting for incoming connections and data, whereas the nature of the client is to solicit data and send it out. IBM has created separate libraries (DLLs) for sending and receiving. Function sets are similar but the underlying behaviors are inherently different.
6. Make sure that **MQCONN** is stored in the start-up file connection establishment function (see **MQCONN** on page 53). **MQCONNX** can be used in place of **MQCONN** and provides additional options when checking compatibility issues.
7. If you are working with only one MQ Manager, the MQ-connect function will be issued only once. However, if you're working with multiple MQ Managers, put this function in the start-up file so you can issue it as needed.
8. Select the MQSeries functions you will use and configure them (see **"MQSeries e*Way External Init Functions" on page 43**)
9. Open a queue using the MQOPEN function (see **MQOPEN** on page 55).
10. Perform an **MQGET** function (see **MQGET** on page 54) or an **MQPUT** function (see **MQPUT** on page 56). This will be determined by whether or not you want to

read from the queue (**get**) or write to the queue (**put**). Two e*Ways must be created to communicate to and from MQSeries.

11 Add an option to close to your finished script.

12 Disconnect from the MQ Manager.

4.1.1. Connecting to a Remote Queue

When accessing an MQSeries remote queue, the INPUT_SHARED option is not allowed. The mq-stdver-eway-functs.monk file must be amended as follows:

In the MQ-stdver-conn-estab function, in the line

```
("MQOO_FAIL_IF_QUIESCING" "MQOO_INPUT_SHARED" "MQOO_OUTPUT"))))
```

remove the parameter "MQOO_INPUT_SHARED" for remote queue access.

4.2 Error Handling

The MQ ***.dll** file checks the execution status of its underlying MQ API functions. In case of a failed operation, this file reports the error code and the reason for the failed call via its internal string variables **MQ-errno** and **MQ-error**.

MQ-errno contains the numeric value associated with the failed call (similar to **errno** for C-library calls). And **MQ-error** is the verbose explanation for the failure.

All the Monk functions in the ***.dll** file have been created to return a Boolean false **(#f),** if the call to an underlying MQ API function that fails. In such cases, examine the contents of **MQ-errno** and **MQ-error** to see what caused the failure.

There are 230 error codes (called reason codes in MQ). Look up these codes in the MQSeries online documentation in the following section:

MQSeries Application Programming Reference, Chapter 5, "Return Codes"

You can take action, as determined by the nature of the error.

Note: *The MQSeries e*Way displays a warning if it receives a message that is larger than the configured message size.*

4.3 Installing the MQSeries (Monk) Sample Schema

The e*Gate Installation CD contains a sample schema, MQ_monk_sample.zip, located in the ../samples/ewmq/ directory. The schema demonstrate a simple scenario using the Monk version of the e*Way Intelligent Adapter for MQSeries.

4.3.1. Install the Sample Schema on the Registry Host

Import the schema at the startup of the e*Gate Schema Designer, or by selecting "New Schema" from the File menu once the e*Gate Schema Designer has opened. For either case, select "Create from export:" and navigate to the .zip file containing the necessary sample.

If the .zip file does not contain a .ctl file, create the following directory :

```
server/registry/repository/schema_name/
```

on your e*Gate registry host, and copy the runtime directory that is contained within the .zip file for the schema, to the newly created directory. When creating the directory, you must use the actual schema name specified when importing the schema.

Chapter 5

MQSeries e*Way Functions

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

- **Basic Functions** on page 39
- **MQSeries e*Way External Init Functions** on page 43
- **MQSeries Monk Functions** on page 50
- **MQSeries Auxiliary Functions** on page 58
- **MQSeries Structure Related Functions** on page 72

In addition, MQ data structures that can be modified using the MQ *,dll files are documented in a table located at:

- **MQSeries Structures** on page 68

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.*

5.1 Basic Functions

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

The basic functions are

event-send-to-egate on page 40

get-logical-name on page 40

send-external-down *on page 41*

send-external-up *on page 41*

shutdown-request on page 41

start-schedule on page 42

stop-schedule on page 42

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	Data to be sent to the e*Gate system.

Return Values

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

Throws

None.

Additional information

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

get-logical-name

Syntax

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

Description

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

Parameters

None.

Return Values

string
Returns the name of the e*Way (as defined by the Schema Designer).

Throws

None.

send-external-down

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown request requests the e*Way to perform the shutdown procedure when there is no outstanding incoming/outgoing event. When the e*Way is ready to act on the shutdown request, in invokes the Shutdown Command Notification Function (see**"Shutdown Command Notification Function" on page 32**). 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.

start-schedule

Syntax

```
(start-schedule)
```

Description

start-schedule requests that the e*Way execute **Exchange Events with External** 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 **Exchange Events with External** specified within the e*Way's configuration file. Execution will be stopped when the e*Way concludes any open transaction. Does not effect any defined schedules, and does not halt the e*Way process itself.

Parameters

None.

Return Values

None.

Throws

None.

5.2 MQSeries e*Way External Init Functions

Access the external init functions from **monk_library\Mq\mq-stdver-eway-funcs.monk**. They provide entry points into the ***.def** file.

The following are examples of default functions for the e*Way's configuration:

MQ-stdver-conn-estab on page 43
MQ-stdver-conn-shutdown on page 44
MQ-stdver-conn-ver on page 44
MQ-stdver-data-exchg on page 45
MQ-stdver-data-exchg-stub on page 45
MQ-stdver-init on page 46
MQ-stdver-neg-ack on page 46
MQ-stdver-pos-ack on page 47
MQ-stdver-proc-outgoing on page 48
MQ-stdver-proc-outgoing-stub on page 48
MQ-stdver-shutdown on page 49
MQ-stdver-startup on page 46

MQ-stdver-conn-estab

Syntax

```
(MQ-stdver-conn-estab)
```

Description

MQ-stdver-conn-estab is used to connect to the external system. This function expects no input.

Parameters

None.

Return Values

This function must return **up** or **success** when the connection is established. Any other return indicates the connection was not established.

Throws

None.

MQ-stdver-conn-shutdown

Syntax

```
(MQ-stdver-conn-shutdown string)
```

Description

MQ-stdver-conn-shutdown is called by the e*Gate system to request that the interface disconnect from the external system, when e*Gate is preparing for a suspend/reload cycle.

Parameters

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

Return Values

string

Any return indicates that the suspend can occur immediately, which puts the e*Way into a **down** state. The user can choose to define the return string as **success**, **#t** (true) or simply a null string.

Throws

None.

MQ-stdver-conn-ver

Syntax

```
(MQ-stdver-conn-ver)
```

Description

MQ-stdver-conn-ver is used to verify the connection with the external system. This function expects no input.

Parameters

None.

Return Values

This function must return **up** or **success** when the connection is established. Any other return indicates the connection was not established.

Throws

None.

MQ-stdver-data-exchg

Syntax

```
(MQ-stdver-data-exchg)
```

Description

MQ-stdver-data-exchg is used to send a message received from the external system to e*Gate. This function expects no input.

Parameters

None.

Return Values

string

This function returns a null string, indicating a successful operation; nothing is sent to e*Gate.

When a **<message-string>** is returned, an Event is sent to e*Gate.

<CONNERR> is used if the connection to the external system was lost, and sets the system state to **down**.

Throws

None.

MQ-stdver-data-exchg-stub

Syntax

```
(MQ-stdver-data-exchg-stub)
```

Description

MQ-stdver-data-exchg-stub s used as a place holder for the function entry point for sending a message received from the external system to e*Gate, when the interface is configured as an outbound only connection. This function is used to catch configuration problems. The function expects no input.

Parameters

None.

Return Values

string

This function returns a null string, indicating a successful operation; nothing is sent to e*Gate.

When a **<message-string>** is returned, an Event is sent to e*Gate.

<CONNERR> is used if the connection to the external system was lost, and sets the system state to **down**.

Throws

None.

MQ-stdver-init

Syntax

```
(MQ-stdver-init)
```

Description

MQ-stdver-init allows you to post Events into the e*Gate system from within a Monk function. This function expects no input.

Parameters

None.

Return Values

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

Throws

None.

MQ-stdver-startup

Syntax

```
(MQ-stdver-startup)
```

Description

MQ-stdver-startup is used for instance specific function loads and environment setup. This function does not expect input.

Parameters

None.

Return Values

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

Throws

None.

MQ-stdver-neg-ack

Syntax

```
(MQ-stdver-neg-ack)
```

Description

MQ-stdver-neg-ack is used to send a negative acknowledgment to the external system, as well as for post processing after failing to send data to e*Gate. This function expects input in the form of data sent to e*Gate.

Parameters

None.

Return Values

string
This function returns a null string, indicating a successful operation.

<CONNERR> indicates a loss of connection with the external. This puts the e*Way into a down state. The e*Way will attempt to connect and upon successful reconnection, the **Negative Acknowledgement** function will be re-executed.

Throws

None.

MQ-stdver-pos-ack

Syntax

```
(MQ-stdver-pos-ack)
```

Description

MQ-stdver-pos-ack is used to send a positive acknowledgment to the external system, as well as for post processing after successfully sending data to e*Gate. This function expects input in the form of data sent to e*Gate.

Parameters

None.

Return Values

string
This function returns null string, indicating a successful operation.

<CONNERR> indicates a loss of connection with the external. This puts the e*Way into a **down** state. The e*Way will attempt to connect and upon successful reconnection the **Positive Acknowledgement** function will be re-executed.

Throws

None.

MQ-stdver-proc-outgoing

Syntax

```
(MQ-stdver-proc-outgoing)
```

Description

MQ-stdver-proc-outgoing is used to send an Event from e*Gate to the external system. This function expects an Event sent from e*Gate as the input.

Parameters

None.

Return Values

string

This function returns a null string, indicating a successful operation.

<RESEND> causes the Event to be immediately resent.

<CONNERR> indicates a loss of connection with the external. This puts the e*Way into a down state. The e*Way will attempt to connect and upon successful reconnection, this function will be re-executed with the same input Event.

<DATAERR> is used if the function experiences problems processing the data. If the function is journaled enabled, the input Event is journaled and the failed Event count is increased.

Throws

None.

Additional Information

Use the **event-send-to-egate** function to place any bad Events into a bad-Event queue.

MQ-stdver-proc-outgoing-stub

Syntax

```
(MQ-stdver-proc-outgoing-stub)
```

Description

MQ-stdver-proc-outgoing-stub is used as a place holder for the function entry point for sending an Event received from e*Gate to the external system, when the interface is configured as an inbound-only connection. This function is used to catch configuration problems. This function expects an Event sent from e*Gate as the input.

Parameters

None.

Return Values

string

This function returns a null string, indicating a successful operation.

<RESEND> causes the Event to be immediately resent.

<CONNERR> indicates a loss of connection with the external. This puts the e*Way into a down state. The e*Way will attempt to connect and upon successful reconnection, this function will be re-executed with the same input Event.

<DATAERR> is used if the function experiences problems processing the data. If the function is journaled enabled, the input Event is journaled and the failed Event count is increased.

Throws

None.

Notes

Use the **event-send-to-egate** function to place any bad Events into a bad-Event queue.

MQ-stdver-shutdown

Syntax

```
(MQ-stdver-shutdown string)
```

Description

MQ-stdver-shutdown is called by the e*Gate system to request that the e*way disconnect from the external system, in preparation for a suspend/reload cycle.

Parameters

Name	Type	Description
string	String	When the e*Way calls this function, e*Gate will pass the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

Any return value indicates that the suspend can occur immediately, which puts the e*Way into a down state.

Throws

None.

5.3 MQSeries Monk Functions

To better understand how the following functions are used, a discussion of the Message Queue Interface (MQI) is needed. The MQI comprises the following items:

- Calls (APIs) through which applications can access the queue manager.
- Structures that applications use to pass data to, and get data from the queue manager.
- Elementary data types for passing data to, and getting data from the queue manager.

The MQSeries Monk functions are

MQBACK on page 50
MQBEGIN on page 51
MQCLOSE on page 51
MQCMIT on page 52
MQCONN on page 53
MQCONNX on page 53
MQDISC on page 54
MQGET on page 54
MQOPEN on page 55
MQPUT on page 56
MQPUT1 on page 57

MQBACK

Syntax

```
(MQBACK conn-handle)
```

Description

MQBACK tells the queue manager to back out of the queue all messages put on the queue or retrieved from the queue as part of a single unit of work.

Parameters

Name	Type	Description
conn-handle	Connection handle	A connection handle.

Return Values

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

Throws

None.

Example

```
(define hConn (MQCONN "Sample_QM"))
(define bool-ret (MQBACK hConn))
)
```

MQBEGIN

Syntax

```
(MQBEGIN conn-handle mqbo-ptr)
```

Description

MQBEGIN starts a unit of work only when MQSeries is acting as an XA transaction coordinator. In all other cases, the start transaction is implicit in **MQCONN**, **MQBACK**, or **MQCMIT**.

Parameters

Name	Type	Description
conn-handle	Connection handle	A connection handle.
mqbo-ptr	String	The definition of the **MQBO** data structure.

Return Values

Boolean

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

Throws

None.

Example

```
(define hConn (MQCONN "Sample_QM"))
(define mqbo-ptr (init-MQBO-struct))
(define bool-ret (MQBEGIN hConn mqbo-ptr))
```

MQCLOSE

Syntax

```
(MQCLOSE conn-hdl obj-hdl option)
```

Description

MQCLOSE closes the queue.

Parameters

Name	Type	Description
conn-hdl	Connection handle	A connection handle.
obj-hdl	Object handle	The object handle.
option	String	The action performed by **MQCLOSE**.

Return Values

Boolean

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

Throws

None.

Examples

```
(define hConn (MQCONN "Sample_QM"))
(define mqod-ptr (init-MQOD-struct))
(define open-option (MQ-create-config-value `#("MQOO_INPUT_SHARED"
"MQOO_OUTPUT" "MQOO_FAIL_IF_QUIESCING")))
(define hObj (MQOPEN hConn mqod-ptr open-option))
(define bool-ret (MQCLOSE hConn hObj 0))
```

MQCMIT

Syntax

```
(MQCMIT conn-handle)
```

Description

MQCMIT commits all changes made to the queue.

Parameters

Name	Type	Description
conn-handle	Connection handle	A connection handle.

Return Values

Boolean

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

Throws

None.

Examples

```
(define hConn (MQCONN "Sample_QM"))
(define bool-ret (MQCMIT hConn))
```

MQCONN

Syntax

```
(MQCONN queue manager name)
```

Description

MQCONN provides a connection to the queue manager.

Parameters

Name	Type	Description
queue manager name	String	The name of the queue manager.

Return Values

string
Returns a connection handle if successful; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(define hConn (MQCONN "Sample_QM") ) where hConn is the connection
handle)
```

MQCONNX

Syntax

```
(MQCONNX queue-manager-name mqcno-ptr)
```

Description

MQCONNX connects the application to the queue manager.

Parameters

Name	Type	Description
queue-manager-name	String	The name of the queue manager.
mqcno-ptr	String	The definition of the **MQCNO** data structure.

Return Values

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

Throws

None.

Examples

```
(define mqcno-ptr (init-MQCNO-struct))
(define hConn (MQCONNX "Sample_QM" mqcno-ptr))
```

Note: *MQCONNX is identical in functionality to MQCONN, but provides the additional parameter for specifying the MQCNO data structure.*

MQDISC

Syntax

```
(MQDISC conn-hdl)
```

Description

MQDISC disconnects the application from the queue manager.

Parameters

Name	Type	Description
conn-hdl	Connection handle	The connection handle.

Return Values

Boolean

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

Throws

None.

Examples

```
(define mqcno-ptr (init-MQCNO-struct))
(define hConn (MQCONNX "Sample_QM" mqcno-ptr))
(define bool-ret (MQDISC hConn))
)
```

MQGET

Syntax

```
(MQGET conn-hdl obj-hdl mqmd-ptr mqgmo-ptr buffer-length)
```

Description

MQGET retrieves the Event.

Parameters

Name	Type	Description
conn-hdl	Connection handle	The connection handle.
obj-hdl	Object handle	The object handle.

Name	Type	Description
mqmd-ptr	String	The definition of the **MQMD** data structure.
mqgmo-ptr	String	The definition of the **MQGMO** data structure.
buffer-length	Integer	The length in bytes of the buffer.

Return Values

string
Returns a message string if successful, otherwise, returns **#f** (false).

Throws

None.

Examples

```
(define mqcno-ptr (init-MQCNO-struct))
(define hConn (MQCONNX "Sample_QM" mqcno-ptr))
(define mqod-ptr (init-MQOD-struct))
(define open-option (MQ-create-config-value '#("MQOO_INPUT_SHARED"
"MQOO_OUTPUT" "MQOO_FAIL_IF_QUIESCING")))
(define hObj (MQOPEN hConn mqod-ptr open-option))
(define mqmd-ptr (init-MQMD-struct))
(define mqgmo-ptr (init-MQGMO-struct))
(define message (MQGET hConn hObj mqmd-ptr mqgmo-ptr 1024))
```

MQOPEN

Syntax

```
(MQOPEN conn-handle mqod-ptr option)
```

Description

MQOPEN opens a message queue.

Parameters

Name	Type	Description
conn-handle	Connection handle	A connection handle.
mqod-ptr	String	The definition of the **MQOD** data structure.
option	String	The action performed by **MQOPEN**.

Return Values

string
Returns an object handle if successful; otherwise, returns **#f** (false).

Throws

None.

Examples

```
((define hConn (MQCONN "Sample_QM"))
(define mqod-ptr (init-MQOD-struct))
(define open-option (MQ-create-config-value '#("MQOO_INPUT_SHARED"
"MQOO_OUTPUT" "MQOO_FAIL_IF_QUIESCING")))
(define hObj (MQOPEN hConn mqod-ptr open-option)))
```

MQPUT

Syntax

```
(MQPUT conn-hdl obj-hdl mqmd-ptr mqpmo-ptr buffer)
```

Description

MQPUT puts the Event in the queue.

Parameters

Name	Type	Description
conn-hdl	Connection handle	The connection handle.
obj-hdl	Object handle	The object handle
mqmd-ptr	String	The definition of the **MQMD** data structure.
mqpmo-ptr	String	The definition of the **MQPMO** data structure.
buffer	String	The data contained in the Event.

Return Values

Boolean

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

Throws

None.

Examples

```
(define mqcno-ptr (init-MQCNO-struct))
(define hConn (MQCONNX "Sample_QM" mqcno-ptr))
(define mqod-ptr (init-MQOD-struct))
(define open-option (MQ-create-config-value '#("MQOO_INPUT_SHARED"
"MQOO_OUTPUT" "MQOO_FAIL_IF_QUIESCING")))
(define hObj (MQOPEN hConn mqod-ptr open-option))
(define mqmd-ptr (init-MQMD-struct))
(define mqpmo-ptr (init-MQPMO-struct))
(define message (MQPUT hConn hObj mqmd-ptr mqpmo-ptr "my testing
message"))
```

MQPUT1

Syntax

```
(MQPUT1 conn-hdl mqod-ptr mqmd-ptr mqpmo-ptr buffer)
```

Description

MQPUT1 puts the Event in the queue. It is functionally equivalent to calling MQOPEN followed by MQPUT, followed by MQCLOSE. The only difference in the syntax for the MQPUT and MQPUT1 calls is that for MQPUT you must specify an object handle, whereas for MQPUT1 you must specify an object descriptor structure (MQOD) as defined in MQOPEN. Use the MQPUT1 call when you want to close the queue immediately after you have put a single message on it.

Parameters

Name	Type	Description
conn-hdl	Connection handle	The connection handle.
mqod-ptr	String	The definition of the **MQOD** data structure.
mqmd-ptr	String	The definition of the **MQMD** data structure.
mqpmo-ptr	String	The definition of the **MQPMO** data structure.
buffer	String	The data contained in the Event.

Return Values

Boolean

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

Throws

None.

Examples

```
(define mqcno-ptr (init-MQCNO-struct))
(define hConn (MQCONNX "Sample_QM" mqcno-ptr))
(define mqod-ptr (init-MQOD-struct))
(define mqmd-ptr (init-MQMD-struct))
(define mqpmo-ptr (init-MQPMO-struct))
(define message (MQPUT1 hConn mqod-ptr mqmd-ptr mqpmo-ptr "my testing
message"))
```

5.4 MQSeries Auxiliary Functions

The functions explained in this section rely on the information presented under "MQSeries Structures" on page 68. The structure names and their field names must be used as input arguments when using these auxiliary functions and must be an exact match of the table entries. The data types for the individual fields specify how they are used in Monk.

The MQSeries Auxiliary Functions are

MQ-check-type on page 58
MQ-configure-close-options on page 59
MQ-configure-get-options on page 59
MQ-configure-open-options on page 61
MQ-configure-options on page 62
MQ-configure-put-options on page 63
MQ-create-config-value on page 64
MQ-get-field on page 64
MQ-init-type on page 65
MQ-reset-type on page 66
MQ-set-field on page 66

MQ-check-type

Syntax

```
(MQ-check-type MQtype MQobject)
```

Description

MQ-check-type checks to see if the **MQobject** is of type **MQtype**.

Parameters

Name	Type	Description
MQtype	String	The string containing the name of an MQ data structure.
MQobject	String	A Monk object used to represent type **MQtype**.

Return Values

Boolean

Returns **#t** (true) if **MQobject** is of type **MQtype**; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(define Object_Descriptor (MQ-init-type "MQOD"))
(define result (MQ-check-type "MQOD" Object_Descriptor))
```

Note: *This function fails if any mismatches appear between the input arguments to this function.*

MQ-configure-close-options

Parameter Name	Description
MQCO_NONE	An option that tells the queue manager no optional close processing is required. This must be specified for ▪ Objects other than queues ▪ Predefined queues ▪ Temporary dynamic queues (but only in those cases where **Hobj** is not the handle returned by the **MQOPEN** call that created the queue) ▪ Distribution lists In all of the above cases, the object is retained and not deleted.
MQCO_DELETE	An option that tells the queue manager to delete the queue. The queue is deleted if either of the following is true: ▪ It is a permanent dynamic queue, and there are no Events on the queue and no uncommitted get or put requests outstanding for the queue (either for the current task or any other task). ▪ It is the temporary dynamic queue that was created by the **MQOPEN** call that returned **Hobj**. In this case, all the Events on the queue are purged.
MQCO_DELETE_ PURGE	An option that tells the queue manager to delete the queue, purging any messages. The queue is deleted if either of the following is true: ▪ It is a permanent dynamic queue and there are no uncommitted get or put requests outstanding for the queue (either for the current task or any other task). ▪ It is the temporary dynamic queue that was created by the **MQOPEN** call that returned **Hobj**.

MQ-configure-get-options

Parameter Name	Description
MQGMO_NONE	An option used when no options are specified.
MQGMO_NO_WAIT	An option that tells the queue manager to return immediately when no acceptable Events are available. This is the opposite of the MQGMO_WAIT option, and is defined to aid program documentation. This is the default if neither option is specified.
MQGMO_WAIT	An option that tells the queue manager to wait for an acceptable Event to arrive. The maximum time for waiting is specified in the **WaitInterval** parameter.

Parameter Name	Description (Continued)
MQGMO_SYNCPOINT	An option that tells the queue manager to get the Event using **syncpoint** control. The request is to operate within the normal unit of work protocols. This marks the Event as unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The Event is made available again after the unit of work is backed out.
MQGMO_NO_ SYNCPOINT	An option that tells the queue manager to get the Event without using **syncpoint** control. The request is to operate within the normal unit of work protocols. The Event is deleted from the queue immediately (unless this is a browse request). The Event can not be made available again by backing out a unit of work.
MQGMO_SET_SIGNAL	An option that requests a signal is to be set. This option is used in conjunction with the **Signal1** and **Signal2** fields to allow applications to proceed with other work while waiting for an Event to arrive, and also (if suitable operating system facilities are available) to wait for Events arriving on more than one queue.
MQGMO_BROWSE_FIRST	An option that tells the queue manager to browse from the beginning of the queue.
MQGMO_BROWSE_NEXT	An option that tells the queue manager to browse from its current position on the queue. The browse cursor is advanced to the next message on the queue that satisfies the selection criteria specified on the **MQGET** call. The message is returned to the application, but remains on the queue.
MQGMO_ACCEPT_ TRUNCATED_MSG	An option that allows truncation of Event data. If the Event buffer is insufficient to hold the complete Event, this option allows the **MQGET** call to fill the buffer with as much of the Event as the buffer can hold, issue a warning completion code, and complete its processing.
MQGMO_MARK_SKIP_ BACKOUT	An option that allows a unit of work to be backed out from the queue, without reinstating the Event marked with this option.
MQGMO_MSG_UNDER_ CURSOR	An option that causes the Event pointed to by the browse cursor to be retrieved, regardless of the **MQMO_*** options specified in the **MatchOptions** field in **MQGMO**. The message is removed from the queue.
MQGMO_LOCK	An option that locks the Event being browsed, to prevent the Event from being visible to any other handle open for the queue. The option can be specified only if one of the following options is also specified: ▪ MQGMO_BROWSE_FIRST ▪ MQGMO_BROWSE_NEXT ▪ MQGMO_BROWSE_MSG_ UNDER_CURSOR
MQGMO_UNLOCK	An option that unlocks the Event. The Event must have been previously locked by an **MQGET** call with the MQGMO_LOCK option.

Parameter Name	Description (Continued)
MQGMO_BROWSE_MSG_ UNDER_CURSOR	An option that causes the Event pointed to by the browser cursor to be retrieved nondestructively, regardless of the MQMO_* options specified in the **MatchOptions** field in **MQGMO**.
MQGMO_SYNCPOINT_IF_ PERSISTENT	An option that tells the queue manager to get the Event using **syncpoint** control if the Event is persistent. A persistent Event has the value MQPER_PERSISTENT in the Persistence field in **MQMD**.
MQGMO_FAIL_IF_ QUIESCING	An option that tells the queue manager to fail when it is in a quiet state.
MQGMO_CONVERT	An option that requests that the application data in the message be converted, to conform to the **CodedCharSetId** and Encoding values specified in the **MsgDesc** parameter on the **MQGET** call, before the data is copied to the buffer parameter.
MQGMO_LOGICAL_ ORDER	An option that controls the order in which Events are returned by successive **MQGET** calls for the queue handle. The option must be specified on each of those calls in order to have an effect.
MQGMO_COMPLETE_ MSG	An option that specifies that only a complete logical Event can be returned by the **MQGET** call.
MQGMO_ALL_MSGS_ AVAILABLE	An option that specifies that Events in a group become available for retrieval only when all Events in the group are available.
MQGMO_ALL_ SEGMENTS_AVAILABLE	An option that specifies that segments in a logical Event become available for retrieval only when all segments in the logical Event are available.

MQ-configure-open-options

Parameter Name	Description
MQOO_INPUT_AS_Q_DEF	An open queue used to get Events using the queue-defined default.
MQOO_INPUT_SHARED	An open queue used to get Events with shared access.
MQOO_INPUT_EXCLUSIVE	An open queue used to get Events with exclusive access.
MQOO_BROWSE	An open queue used to browse messages. The queue is opened for use with subsequent **MQGET** calls with one of the following options: ▪ MQGMO_BROWSE_ FIRST ▪ MQGMO_BROWSE_ NEXT ▪ MQGMO_BROWSE_MSG_UNDER_CURSOR
MQOO_OUTPUT	An open queue to put Events into.

Parameter Name	Description
MQOO_SAVE_ALL_CONTEXT	A Save context used when an Event is retrieved. Context information is associated with this queue handle.
MQOO_PASS_IDENTITY_ CONTEXT	An option which allows **MQPMO_PASS_ IDENTITY_CONTEXT** option to be specified in the **PutMsgOpts** parameter, which allows all identity context to be passed when an Event is put on a queue.
MQOO_PASS_ALL_CONTEXT	An option which allows **MQPMO_PASS_ ALL_CONTEXT** option to be specified in the **PutMsgOpts** parameter, which allows all context to be passed when an Event is put on a queue.
MQOO_SET_IDENTITY_ CONTEXT	An option which allows **MQPMO_SET_ IDENTITY_CONTEXT** option to be specified in the **PutMsgOpts** parameter, which allows the identity context to be set when an Event is put on a queue.
MQOO_SET_ALL_CONTEXT	An option which allows the **MQPMO_SET_ALL_CONTEXT** option to be specified in the **PutMsgOpts** parameter, which allows all context to be set when an Event is put on a queue.
MQOO_ALTERNATE_USER_ AUTHORITY	A method to validate that the **AlternateUserId** field in the **ObjDesc** parameter contains a user identifier.
MQOO_FAIL_IF_QUIESCING	An option that tells the queue manager to fail when it is in a quiet state (inactive).

MQ-configure-options

Syntax

```
(MQ-configure-options options)
```

Description

MQ-configure-options merges the long integer values in the passed-in vector into one long integer-option value.

Parameters

Name	Type	Description
options	Integer	The action of **MQ-configure-options** is controlled with this function.

Return Values

integer
Returns an option value consisting of a long integer.

Boolean
Returns **#f** (false) if unsuccessful.

Throws

None.

Examples

```
(define INPUT_SHARED 2)
(define OUTPUT 16)
(define FAIL_IF_QUIESCING 8192)
(MQ-configure-options `#(INPUT_SHARED OUTPUT FAIL_IF_QUIESCING))
```

MQ-configure-put-options

Parameter Name	Description
MQPMO_NONE	An option used when no options are specified.
MQPMO_SYNCPOINT	An option that tells the queue manager to put the Event into the queue using **syncpoint** control. The request is to operate within the normal unit of work protocols. The Event is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the Event is deleted.
MQPMO_NO_SYNCPOINT	An option that tells the queue manager to put the Event into the queue without using **syncpoint** control. The request is to operate outside the normal unit of work protocols. The Event is available immediately and can not be deleted by backing out a unit of work.
MQPMO_DEFAULT_CONTEXT	An option that tells the queue manager to use default context. An Event is to have default context information associated with it, for both identity and origin.
MQPMO_NEW_MSG_ID	An option that causes the queue manager to replace the contents of the **MsgId** field in **MQMD** with a new identifier.
MQPMO_NEW_CORREL_ID	An option that causes the queue manager to replace the contents of the **CorrelId** field in **MQMD** with a new identifier.
MQPMO_PASS_IDENTITY_ CONTEXT	An option used to pass the identity context from an input queue handle.
MQPMO_PASS_ALL_CONTEXT	An option used to pass all context from an input queue handle.
MQPMO_SET_IDENTITY_ CONTEXT	An option used to set identity context from the application.
MQPMO_SET_ALL_CONTEXT	An option used to set all context from the application.
MQPMO_ALTERNATE_USER_ AUTHORITY	An option that tells the queue manager to validate the Event using a specified user identifier.
MQPMO_FAIL_IF_QUIESCING	An option that tells the queue manager to fail if the queue manager is quiet (that is, an inactive state).
MQPMO_NO_CONTEXT	An option that tells the e*Way that no context is to be associated with the Event.

Parameter Name	Description
MQPMO_LOGICAL_ORDER	An option that tells the queue manager in what order the application will put Events in groups and segments of logical Events.

MQ-create-config-value

Syntax

```
(MQ-create-config-value options )
```

Description

MQ-create-config-value converts the strings in the input vector to integer equivalents in a vector of integers and passes that vector to the **MQ-configure-options** function.

Parameters

Name	Type	Description
options	String	A vector of strings to be converted to integers.

Return Values

integer
Returns an option value consisting of a long integer.

Throws

None.

Examples

```
(define MQ-create-config-value
 (lambda (values)
  (let ((options 0)(len 0))
   (set! len (vector-length values))
   (set! options (make-vector len 0))
   (do ((i 0 (+ i 1))) ((>= i len))
    (vector-set! options i (cadr (assoc (string->symbol (vector-ref
values i)) $MQ-ASSOCS$)))
   )
   (MQ-configure-options options)
)))
```

Note: *This function fails if a string that is not a part of the supported list is passed to it.*

MQ-get-field

Syntax

```
(MQ-get-field MQtype field_name MQobject)
```

Description

MQ-get-field returns the value of the specified field in **MQobject**.

Parameters

Name	Type	Description
MQtype	String	The string containing the name of an MQ-data structure.
field_name	String	A Monk object used to represent type **MQtype**.
MQobject	String	A Monk object used to represent type **MQtype**.

Return Values

string
Returns the basic MQ-data-type of the field.

Throws

None.

Examples

```
(define MQGMO (MQ-init-type "mqgmo"))
(define GetOption (MQ-create-config-value `#("MQGMO_SYNCPOINT"))
(MQ-set-field "MQGMO" "Options" MQGMO GetOption)
(define value (MQ-get-field "MQGMO" "Options" MQGMO))
```

Note: *This function fails if any mismatches appear between the input arguments to this function.*

MQ-init-type

Syntax

```
(MQ-init-type MQtype )
```

Description

MQ-init-type creates the specified **MQobject** and initializes it with default values.

Parameters

Name	Type	Description
MQtype	String	The string containing the name of an MQ-data structure.

Return Values

string
Returns an **MQobject** of type string.

Throws

None.

Examples

```
(define Object_Descriptor (MQ-init-type "MQOD"))
```

Note: *This function fails if the input argument to this function is not supported by the* ***monk_MQclient.dll*** *or* ***monk_MQserver.dll*** *files.*

MQ-reset-type

Syntax

```
(MQ-reset-type MQtype MQobject)
```

Description

MQ-reset-type takes an **MQobject** of type **MQtype** and resets its contents to their default values.

Parameters

Name	Type	Description
MQtype	String	The string containing the name of an MQ-data structure.
MQobject	String	A Monk object used to represent type **MQtype**.

Return Values

Boolean
Returns **#t** (true) if **MQobject** is of type **MQtype**; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(define Object_Descriptor (MQ-init-type "MQOD"))
...
(MQ-reset-type "MQOD" Object_Descriptor)
```

Note: *This function fails if any mismatches appear between the input arguments to this function.*

MQ-set-field

Syntax

```
(MQ-set-field MQtype field_name MQobject value)
```

Description

MQ-set-field takes an **MQobject** of type **MQtype** and sets the value of its internal field (as specified in **field_name**) to the given value.

Parameters

Name	Type	Description
MQtype	String	The string containing the name of an MQ-data structure.
field_name	String	A string containing the name of the member field in **MQtype**.
MQobject	String	A Monk object used to represent type **MQtype**.
value	String	The value assigned to **field_name**.

Return Values

Boolean

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

Throws

None.

Examples

```
(define MQGMO (MQ-init-type "mqgmo"))
(define GetOption (MQ-create-config-value '#("MQGMO_SYNCPOINT"))
(MQ-set-field "MQGMO" "Options" MQGMO GetOption)
```

Note: *This function fails if any mismatches appear between the input arguments to this function.*

5.5 MQSeries Structures

The table, starting below, documents the MQ data structures that the MQ ***.dll** files allow you to modify. The first column contains the name of the structure, the second column shows a subset of the fields within that structure, which you can modify, and the third column shows the basic data types for the individual fields.

Structure	Field Name	Data Type
MQBO	Options Version	MQLONG MQLONG
MQCNO	Options Version	MQLONG MQLONG
MQDH	CodedCharSetId Encoding Flags Format ObjectRecOffset PutMsgRecFields PutMsgRecOffset RecsPresent StrucLength Version	MQLONG MQLONG MQLONG STRING MQLONG MQLONG MQLONG MQLONG MQLONG MQLONG
MQDLH	CodedCharSetId DestQMgrName DesQName Encoding Format PutApplName PutApplType PutDate PutTime Reason Version	MQLONG STRING STRING MQLONG STRING STRING STRING STRING STRING MQLONG MQLONG
MQGMO	GroupStatus MatchOptions Options Reserved1 ResolvedQName SegmentStatus Segmentation Signa11 Signa12 Version WaitInterval	MQCHAR MQLONG MQLONG MQCHAR STRING MQCHAR MQCHAR MQLONG MQLONG MQLONG MQLONG

Structure (Continued)	Field Name	Data Type
MQIIH	Authenticator	STRING
	CodedCharSetId	MQLONG
	CommitMode	MQCHAR
	Encoding	MQLONG
	Flags	MQLONG
	Format	STRING
	LTermOverride	STRING
	MFSMapName	STRING
	ReplyToFormat	STRING
	Reserved	MQCHAR
	SecurityScope	MQCHAR
	StrucLength	MQLONG
	TranInstanceId	STRING
	TranState	MQCHAR
	Version	MQLONG
MQMD	AccountingToken	STRING
	ApplIdentityData	STRING
	ApplOriginData	STRING
	BackoutCount	MQLONG
	CodedCharSetId	MQLONG
	CorrelId	STRING
	Encoding	MQLONG
	Expiry	MQLONG
	Feedback	MQLONG
	Format	STRING
	GroupId	STRING
	MsgFlags	MQLONG
	MsgId	STRING
	MsgSeqNumber	MQLONG
	MsgType	MQLONG
	Offset	MQLONG
	OriginalLength	MQLONG
	Persistence	MQLONG
	Priority	MQLONG
	PutApplName	STRING
	PutApplType	MQLONG
	PutDate	STRING
	PutTime	STRING
	ReplyToQ	STRING
	ReplyToQMgr	STRING
	Report	MQLONG
	UserIdentifier	STRING
	Version	MQLONG

Structure (Continued)	Field Name	Data Type
MQMD1	AccountingToken ApplIdentityData ApplOriginData BackoutCount CodedCharSetId CorrelId Encoding Expiry Feedback Format	STRING STRING STRING MQLONG MQLONG STRING MQLONG MQLONG MQLONG STRING
	MsgId MsgType Persistence Priority PutApplName PutApplType PutDate PutTime ReplyToQ ReplyToQMgr Report UserIdentifier Version	STRING MQLONG MQLONG MQLONG STRING MQLONG STRING STRING STRING STRING MQLONG STRING MQLONG
MQMDE	CodedCharSetId Encoding Flags Format GroupId MsgFlags MsgSeqNumber Offset OriginalLength StrucLength Version	MQLONG MQLONG MQLONG STRING STRING MQLONG MQLONG MQLONG MQLONG MQLONG MQLONG
MQOD	AlternateUsrId DynamicQName InvalidDestCount KnownDestCount ObjectName ObjectQMgrName ObjectRecOffset ObjectType RecsPresent ResponseRecOffset UnknownDestCount Version	STRING STRING MQLONG MQLONG STRING STRING MQLONG MQLONG MQLONG MQLONG MQLONG MQLONG
MQOR	ObjectName ObjectQMgrName	STRING STRING

Structure (Continued)	Field Name	Data Type
MQPMO	Context	MQLONG
	InvalidDestCount	MQLONG
	KnownDestCount	MQLONG
	Options	MQLONG
	Put6MsgRecFields	MQLONG
	PutMsgRecOffset	MQLONG
	RecsPresent	MQLONG
	ResolvedQMgrName	STRING
	ResolvedQName	STRING
	ResponseRecOffset	MQLONG
	Timeout	MQLONG
	UnknownDestCount	MQLONG
	Version	MQLONG
MQRMH	CodedCharSetId	MQLONG
	DataLogicalLength	MQLONG
	DataLogicalOffset	MQLONG
	DataLogicalOffset2	MQLONG
	DestEnvLength	MQLONG
	DestEnvOffset	MQLONG
	DestNameLength	MQLONG
	DestNameOffset	MQLONG
	Encoding	MQLONG
	Flags	MQLONG
	Format	STRING
	ObjectInstanceId	STRING
	ObjectType	STRING
	SrcEnvLength	MQLONG
	SrcEnvOffset	MQLONG
	SrcNameLength	MQLONG
	SrcNameOffset	MQLONG
	StrucLength	MQLONG
	Version	MQLONG
MQRR	CompCode	MQLONG
	Reason	MQLONG
MQTM	ApplId	STRING
	ApplType	MQLONG
	EnvData	STRING
	ProcessName	STRING
	QName	STRING
	TriggerData	STRING
	UsrData	STRING
	Version	MQLONG

Structure (Continued)	Field Name	Data Type
MQTMC2	ApplId ApplType EnvData ProcessName QMgrName QName Trigger Data UsrData Version	STRING STRING STRING STRING STRING STRING STRING STRING STRING
MQXQH	MsgDesc RemoteQMgrName RemoteQName Version	MQMD1 STRING STRING MQLONG

5.6 MQSeries Structure Related Functions

MQMD-struct-get-Contents

Syntax

```
(MQMD-struct-get-Contents MQMDpointer)
```

Description

MQMD-struct-get-Contents creates a binary image of all the fields in one action.

Parameters

Name	Type	Description
MQMDpointer	Opaque handle	Handle or pointer to the MQMD structure.

Return Values

blob

Returns the binary image of all fields.

Throws

None.

Additional Information

The MQMD structure is composed of several fields. In order to grab an image of the MQMD (and avoid doing multiple "gets" and "puts") use the function **MQMD-struct-get-Contents**, which provides a binary image of all the fields in one action. The inverse function, **MQMD-struct-set-Contents** takes this binary image and superimposes it on the outgoing message. (These same functions exists for every supported MQ structure

that the .dll supports; XXX-struct-get-Contents or XXX-struct-set-Contents, substituting the structure name for XXX.)

MQMD-struct-set-Contents

Syntax

```
(MQMD-struct-set-contents MQMDpointer blob)
```

Description

MQMD-struct-set-Contents takes the binary image created by MQMD-struct-set-Contents and superimposes it on the outgoing message.

Parameters

Name	Type	Description
MQMDpointer	Opaque handle	Handle or pointer to the MQMD structure.
blob	Blob	The binary image that is being set.

Return Values

Boolean

Returns **#t** (true)

Throws

None.

Additional Information

The MQMD structure is composed of several fields. In order to grab an image of the MQMD (and avoid doing multiple "gets" and "puts") use the function **MQMD-struct-get-Contents**, which provides a binary image of all the fields in one action. The inverse function, **MQMD-struct-set-Contents** takes this binary image and superimposes it on the outgoing message. (These same functions exists for every supported MQ structure that the .dll supports; XXX-struct-get-Contents or XXX-struct-set-Contents, substituting the structure name for XXX.)

Index

A

Additional Path parameter **26**
AIX **8**
  patch **11**
Auxiliary Library Directories parameter **26**

C

components **7**
configuration parameters **13**
  Additional Path **26**
  Auxiliary Library Directories **26**
  Down Timeout **16**
  Exchange Data Interval **16**
  Exchange Data With External Function **28**
  External Connection Establishment Function **29**
  External Connection Shutdown Function **30**
  External Connection Verification Function **30**
  Forward External Errors **14**
  Journal File Name **14**
  Max Failed Messages **14**
  Max Resends Per Message **14**
  Monk Environment Initialization File **27**
  Negative Acknowledgment Function **31**
  Positive Acknowledgement Function **31**
  Process Outgoing Message Function **28**
  Resend Timeout **17**
  Shutdown Command Notification Function **32**
  Start Exchange Data Schedule **16**
  Startup Function **27**
  Stop Exchange Data Schedule **15**
  Up Timeout **16**
  Zero Wait Between Successful Exchanges **17**
connection test queue name **34**

D

Down Timeout parameter **16**

E

enable connection test **34**
environment variables
  Client mode **34**
  for UNIX **34**
  for Windows **34**
error handling **37**
Exchange Data Interval parameter **16**
Exchange Data with External Function parameter **28**
External Connection Establishment Function parameter **29**
External Connection Shutdown Function parameter **30**
External Connection Verification Function parameter **30**
external system requirements **8**

F

Forward External Errors parameter **14**
Functions
  get-logical-name **40**
  start-schedule **42**
  stop-schedule **42**
functions **57**
  MQBACK **50**
  MQBEGIN **51**
  MQ-check-type **58**
  MQCLOSE **51**
  MQCMIT **52**
  MQ-configure-close-options **59**
  MQ-configure-get-options **59**
  MQ-configure-open-options **61**
  MQ-configure-options **62**
  MQ-configure-put-options **63**
  MQCONN **53**
  MQCONNX **53**
  MQ-create-config-value **64**
  MQDISC **54**
  MQGET **54**
  MQ-get-field **64**
  MQ-init-type **65**
  MQOPEN **55**
  MQPUT **56**
  MQ-reset-type **66**
  MQSeries **50**
  MQSeries auxiliary **58**
  MQSeries structures **68**
  MQ-set-field **66**
  MQ-stdver-conn-estab **43**
  MQ-stdver-conn-shutdown **44**
  MQ-stdver-data-exchg **45**
  MQ-stdver-data-exchg-stub **45**
  MQ-stdver-init **46**
  MQ-stdver-neg-ack **46**
  MQ-stdver-pos-ack **47**
  MQ-stdver-proc-outgoing **48**
  MQ-stdver-proc-outgoing-stub **48**

MQ-stdver-shutdown **49**
MQ-stdver-startup **46**

G

get-logical-name **40**

I

implementation notes **36**
installation **9**
AIX **11**
Lotus Notes sample schema **37**
installation procedure
UNIX **10**
Windows NT and Windows 2000 **9**
intended reader **7**

J

Journal File Name parameter **14**

L

local MQ install **33**

M

Max Failed Messages parameter **14**
Max Resends Per Message parameter **14**
message queue **32**
Monk Environment Initialization File parameter **27**
MQ get buffer length **33**
MQ settings parameter **32**
MQBACK **50, 51**
MQBEGIN **51**
MQ-check-type **58**
MQCLOSE **51**
MQCMIT **52**
MQ-configure-close-options **59**
MQ-configure-get-options **59**
MQ-configure-open-options **61**
MQ-configure-options **62**
MQ-configure-put-options **63**
MQCONN **53**
MQCONNX **53**
MQ-create-config-value **64**
MQDISC **54**
MQGET **54**
MQ-get-field **64**
MQ-init-type **65**
MQOPEN **55**
MQPUT **56**
MQPUT1 **57**
MQ-reset-type **66**
MQSeries auxiliary functions **58**
MQSeries DataGateWay
set up **36**
MQSeries monk functions **50**
MQSeries structures **68**
MQ-set-field **66**
MQ-stdver-conn-estab **43**
MQ-stdver-conn-shutdown **44**
MQ-stdver-data-exchg **45**
MQ-stdver-data-exchg-stub **45**
MQ-stdver-init **46**
MQ-stdver-neg-ack **46**
MQ-stdver-pos-ack **47**
MQ-stdver-proc-outgoing **48**
MQ-stdver-proc-outgoing-stub **48**
MQ-stdver-shutdown **49**
MQ-stdver-startup **46**

N

Negative Acknowledgment Function parameter **31**

O

Overview **6**

P

parameters
connection test queue **34**
enable connection test **34**
local MQ install **33**
MQ get buffer length **33**
MQ settings **32**
queue manager name **33**
queue name **33**
Positive Acknowledgment Function parameter **31**
Process Outgoing Message Function parameter **28**

Q

queue manager **33**
queue manager name **33**
queue name **33**

R

remote queue
connecting to **37**
necessary modifications **37**
Resend Timeout parameter **17**

S

Schedules **42**
Shutdown Command Notification Function parameter **32**
Start Exchange Data Schedule parameter **16**
start-schedule **42**
Startup Function parameter **27**
stcewgenericmonk.exe **7**
Stop Exchange Data Schedule parameter **15**
stop-schedule **42**
system requirements **7**
    AIX **8**
    external **8**

U

UNIX
    installation **10**
Up Timeout parameter **16**

W

Windows **9**

Z

Zero Wait Between Successful Exchanges parameter **17**