SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Lotus Notes User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Version 20050504125637.

Contents

Chapter 1

Introduction 6

Overview 6
Intended Reader 6
Components 6

Supported Operating Systems 7

System Requirements 7
External System Requirements 7

Chapter 2

Installation 8

Windows Installation 8
Pre-installation 8
Installation Procedure 8

Files/Directories Created by the Installation 9

Chapter 3

Configuration 11

e*Way Configuration Parameters 11
General Settings 11
Journal File Name 12
Max Resends Per Message 12
Max Failed Messages 12
Forward External Errors 12
Communication Setup 13
Start Exchange Data Schedule 13
Stop Exchange Data Schedule 14
Exchange Data Interval 14
Down Timeout 14
Up Timeout 14
Resend Timeout 15
Zero Wait Between Successful Exchanges 15
Monk Configuration 15
Operational Details 17

How to Specify Function Names or File Names 23
Additional Path 24
Auxiliary Library Directories 24
Monk Environment Initialization File 25
Startup Function 25
Process Outgoing Message Function 26
Exchange Data with External Function 26
External Connection Establishment Function 27
External Connection Verification Function 28
External Connection Shutdown Function 28
Positive Acknowledgment Function 28
Negative Acknowledgment Function 29
Shutdown Command Notification Function 30
Lotus Notes Settings 30
Lotus Notes Server Name 30
Password for the Notes Server 30
Environment Configuration 31
External Configuration Requirements 31

Chapter 4
Implementation 32
Functional Overview of the Sample Schema 32
Installing the Lotus Notes Sample Schema 33
Install the Sample Schema on the Registry Host 33
Files Included with the Sample Schema 33

Chapter 5
Lotus Notes e*Way Functions 35
Basic Functions 35
event-send-to-egate 36
get-logical-name 37
send-external-down 38
send-external-up 39
shutdown-request 40
start-schedule 41
stop-schedule 42
Lotus Notes Functions 43
General Usage Notes 43
DBHandleOK 45
GetDBHandleStatus 46
GetFieldData 47
GetFieldList 48
GetNotHandleByUNID 49
LNAck 50
LNConnect 51
LNExchange 52
LNGetDBHandle 53
LNInit 54

LNNak 55
LNNotesRead 56
LNNotify 57
LNOutgoing 58
LNShutdown 59
LNStartup 60
LNVerify 61
MarkAsResponse 62
MarkNoteAsRead 63
MarkNoteAsUnRead 64
NextNoteHandle 65
NIFCloseCollection 66
NIFOpenCollection 67
NoteIDAsString 68
NoteHasParent 69
NotesInit 70
NotesTerm 71
NSFDbClose 72
NSFDbOpen 73
NSFDbOpenNet 74
NSFItemAppendTextList 75
NSFItemGetTextListEntries 76
NSFItemGetTextListEntry 77
NSFItemInfo 78
NSFItemSetNumber 79
NSFItemSetText 80
NSFNoteClose 81
NSFNoteCreate 82
NSFNoteDelete 83
NSFNoteUpdateExtended 84
OpenNoteByID 85
SendMail 86

Index 88

Chapter 1

Introduction

This chapter includes an overview of SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way Intelligent Adapter for Lotus Notes, the components that make up the e*Way, and the system requirements for installing the e*Way.

1.1 Overview

Lotus Notes is an integrated environment that provides users with the ability to access and manage many types of information including e-mail, calendar of appointments, personal contacts and to-dos as well as Web pages, News Groups and intranet applications

The Lotus Notes e*Way enables the e*Gate system to access data from a Lotus Notes, database. The e*Way can act as an inbound or outbound e*Way depending on its configuration. An inbound e*Way brings files into the e*Gate environment, queuing Events. Similarly, an outbound e*Way populates a Lotus Notes database.

The Lotus Notes e*Way supports

- Remote or local database access
- Sending of e-mail messages
- Direct access by View or UNID
- Sequential access to Documents according to a View

This document describes how to install and configure the Lotus Notes e*Way.

1.1.1. Intended Reader

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

1.1.2. Components

The Lotus Notes e*Way comprises the following:

- **stcewgenericmonk.exe**, the executable component

- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk function scripts
- Library files

A complete list of installed files appears in **Table 1 on page 10**.

1.2 Supported Operating Systems

The Lotus Notes e*Way is available on the following operating system:

- Windows 2000 and Windows XP

1.3 System Requirements

To use the Lotus Notes 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.

1.3.1. External System Requirements

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

- There must be a Lotus Notes Client 4.6.2 installation on the same host as the e*Gate Participating Host.
- The special DLL used for password event handling must be placed in the Lotus Notes Client area.

Chapter 2

Installation

This chapter explains how to install the Lotus Notes e*Way.

2.1 Windows Installation

2.1.1. Pre-installation

Lotus Notes 4.6.2a Client and the ID file must be installed on the system that you are going to install the e*Way on, as follows:

- You must also have a valid user name and password for Lotus Notes.
- The Lotus Notes installation directory must be on the environment path.
- If a Lotus Notes server is installed on the system that you are going to install the e*Way on, shut down the server before running the setup program; otherwise, InstallShield may not operate correctly.
- Exit all Windows programs before running the setup program, including any anti-virus applications.
- You must have Administrator privileges to install this e*Way.

2.1.2. Installation Procedure

To install the Lotus Notes e*Way on a Windows system

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

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

5 After the installation is complete, exit the install utility.

6 Copy the file **stc_notesextmgr.dll** from the system running e*Gate to your Lotus Notes installation.

The Windows default directory for finding the file is:

c:\eGate\server\registry\repository\default\bin\win32

The default Lotus Notes directory is:

c:\notes

7 Edit the **notes.ini** file. This is usually found in the **\winnt** directory.

Add or edit the following line:

```
EXTMGR_ADDINS=stc_notesextmgr
```

Save the changes and exit **notes.ini**.

8 Launch the Schema Designer.

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

10 Display the new e*Way's properties.

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

12 Select the file **stcewgenericmonk.exe**.

13 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 Files/Directories Created by the Installation

The Lotus Notes e*Way installation process will install 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

e*Gate Client Directory	File(s)
bin\	stc_monklnotes.dll stc_notesextmgr.dll
configs\stcewgenericmonk\	LotusNotes.def
monk_library\	lnotes.gui
monk_library\ewlnotes\	LNAck.monk LNConnect.monk LNExchange.monk LNGetDBHandle.monk LNInit.monk LNNak.monk LNNotesRead.monk LNNotesWriteByID.monk LNNotify.monk LNOutgoing.monk LNShutdown.monk LNStartup.monk LNVerify.monk

Chapter 3

Configuration

The Lotus Notes e*Way must be configured before use. This chapter lists all the configuration parameters used by the e*Way together with all supporting information needed, including Monk configuration for connection to the external system. Also provided are the parameters that control access to a Lotus Notes database.

3.1 e*Way Configuration Parameters

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

To change e*Way configuration parameters:

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

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

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

- General Settings
- Communication Setup
- Monk Configuration
- Lotus Notes 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" on page 12**.
- When its receipt is due to an external error, but Forward External Errors is set to **No**. (See **"Forward External Errors" on page 12** for more information.)

Max Resends Per Message

Description

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

Required Values

An integer between 1 and 1,000,000. 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 1,000,000. 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 26** for more information.

Required Values

Yes or **No**. The default value, **Yes**, specifies that error messages will be forwarded.

See **"Schedule-driven Data Exchange Functions" on page 20** 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 Acknowledgement** 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 26**, **"Exchange Data Interval" on page 14**, and **"Stop Exchange Data Schedule" on page 14** 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 14** and **"Stop Exchange Data Schedule" on page 14** 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 27** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Up Timeout

Description

Specifies the number of seconds the e*Way will wait between calls to the **External Connection Verification** function. See **"External Connection Verification Function" on page 28** for more information.

Required Values

An integer between 1 and 86,400. The default is 15.

Resend Timeout

Description

Specifies the number of seconds the e*Way will wait between attempts to resend 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**.

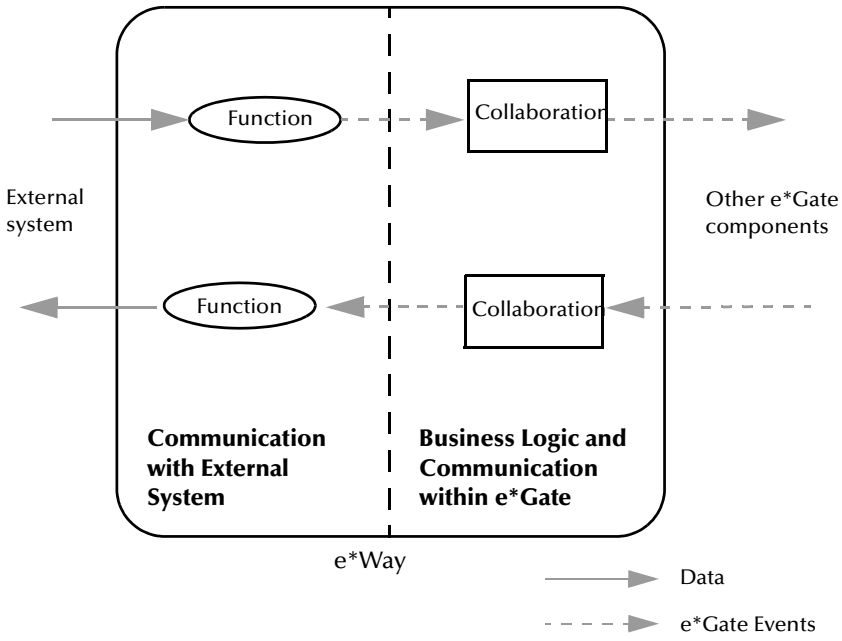
See **"Exchange Data with External Function" on page 26** for more information.

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 16**) 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**).

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.

Operational Details

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

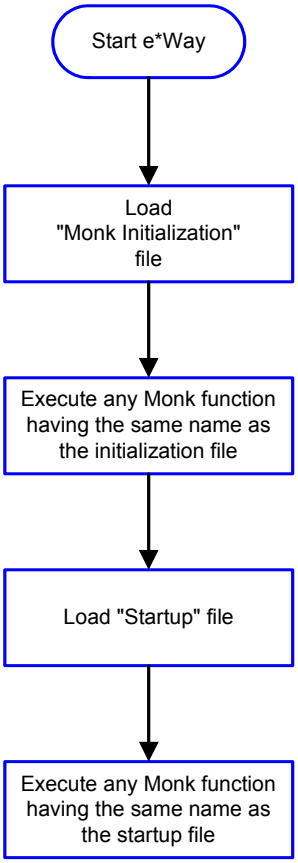
Type of Operation	Name
Initialization	**Startup Function** on page 25 (also see **Monk Environment Initialization File** on page 25)
Connection	**External Connection Establishment Function** on page 27 **External Connection Verification Function** on page 28 **External Connection Shutdown Function** on page 28
Schedule-driven data exchange	**Exchange Data with External Function** on page 26 **Positive Acknowledgment Function** on page 28 **Negative Acknowledgment Function** on page 29
Shutdown	**Shutdown Command Notification Function** on page 30
Event-driven data exchange	**Process Outgoing Message Function** on page 26

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

Initialization Functions

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

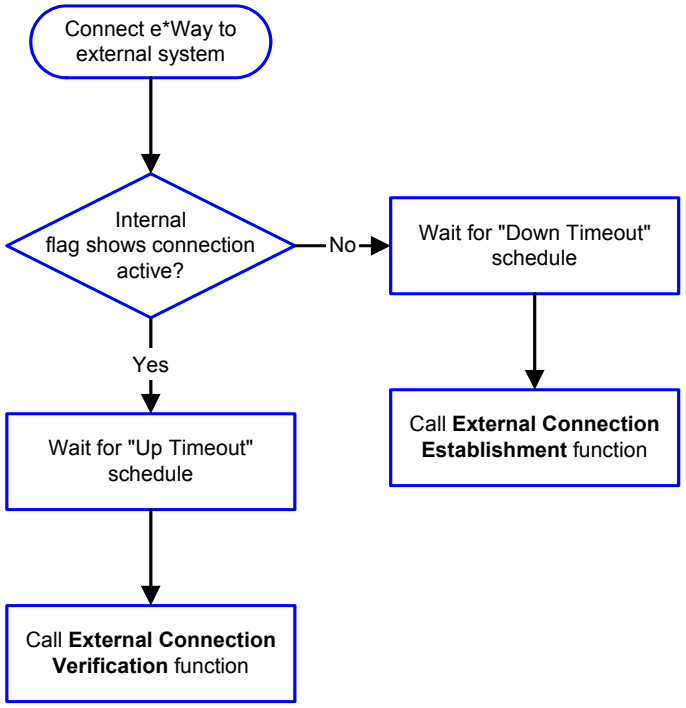
Figure 2 Initialization Functions



Connection Functions

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

Figure 3 Connection Establishment and Verification Functions

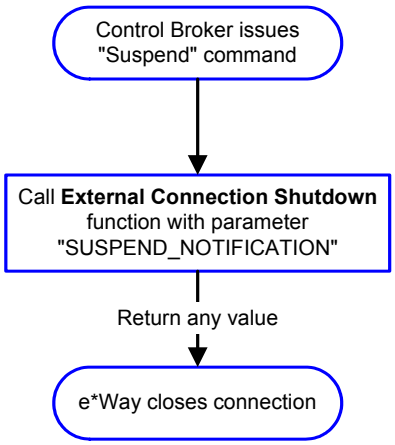


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 5 on page 21** *and* **Figure 7 on page 23** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 39 and* **send-external-down** *on page 38 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

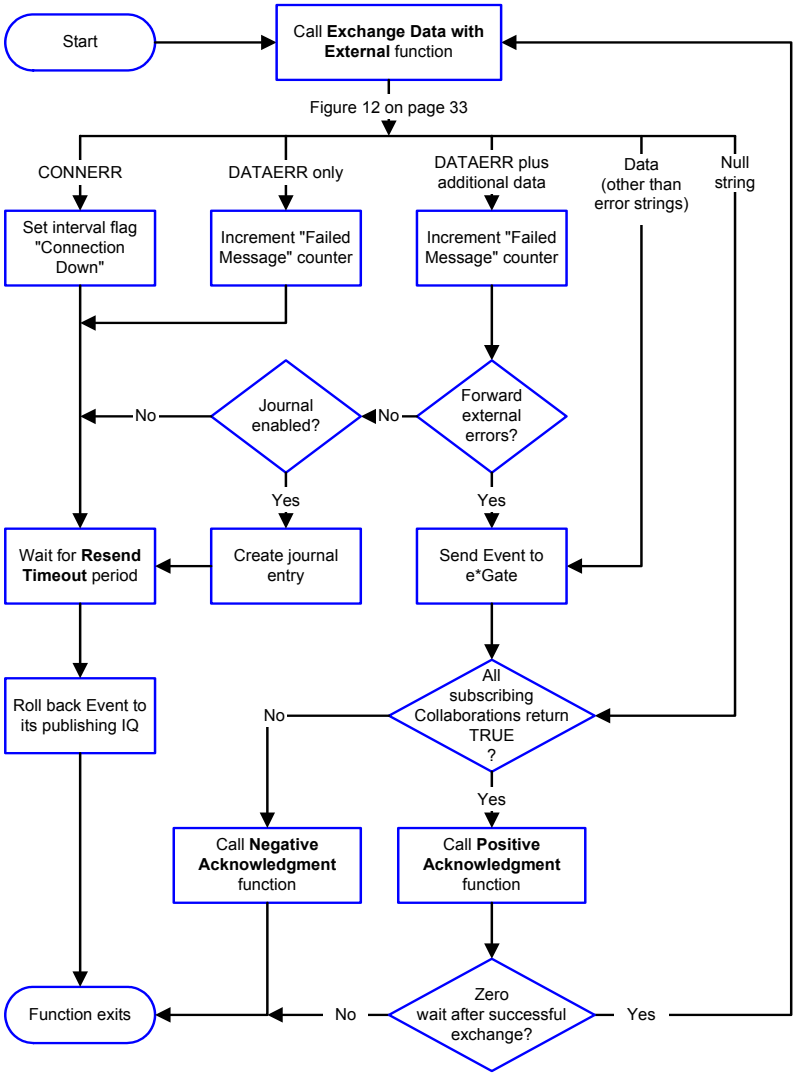
Figure 5 on page 21 illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

"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.

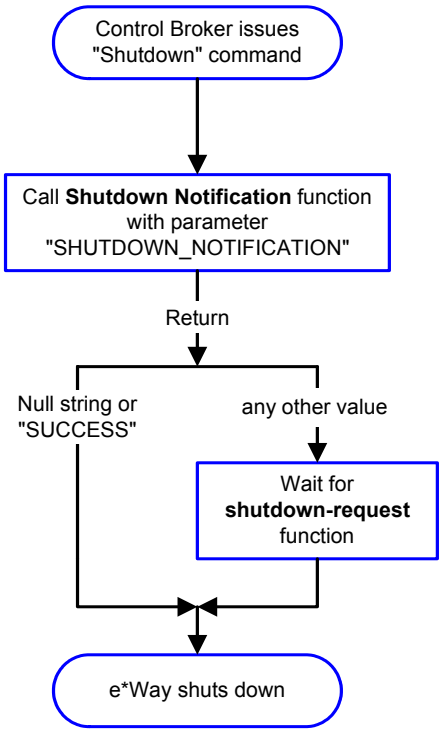
Figure 5 Schedule-driven Data Exchange Functions



Shutdown Functions

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

Figure 6 Shutdown Functions



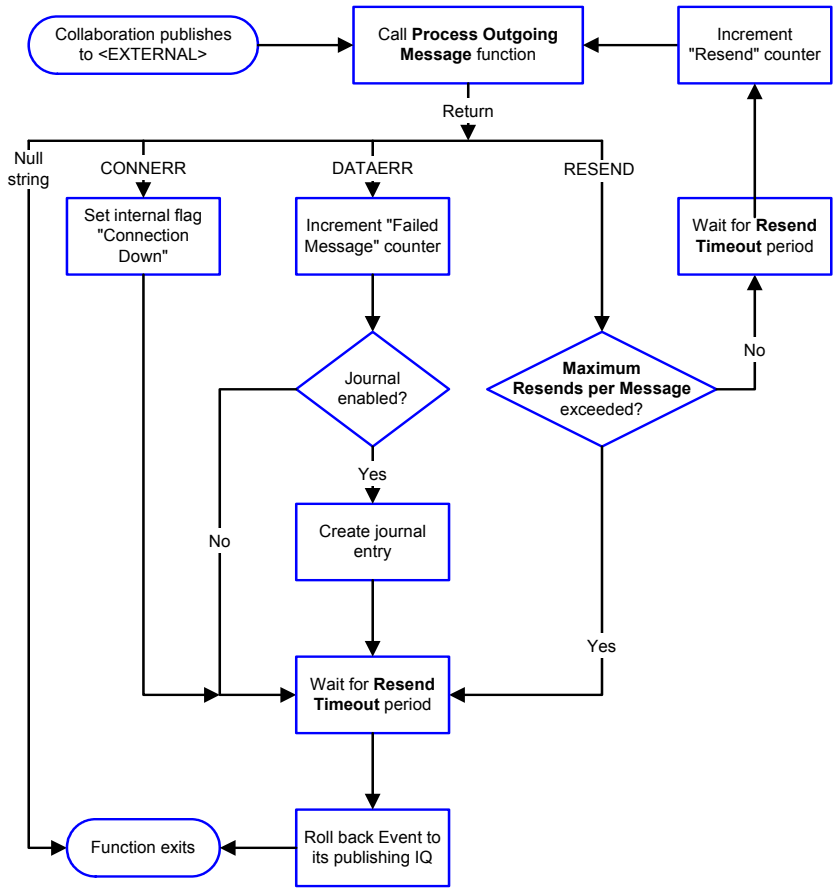
Event-driven Data Exchange Functions

Figure 7 below 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/ewlnotes**.

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 24** 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 **LNInit**.

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 2 on page 18**).

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 default is **LNStartup.**

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

Process Outgoing Message Function

Description

Specifies the Monk function responsible for sending outgoing 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 e*Way invokes this function when one of its Collaborations publishes an Event to an **<EXTERNAL>** destination, as specified within the Schema Designer (see **Figure 7 on page 23** for more details).

This function requires a non-null string as input (the outgoing Event to be sent) and must return a string, as follows:

- 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 those in the previous list 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.

The default is **LNOutgoing.**

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 appropr0iate IQs) configured to process those Events. See* **event-send-to-egate *on page 36*** *for more information.*

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven). See **Figure 5 on page 21** for more details.

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, as follows:

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

This function is initially triggered by the **Start Data Exchange** schedule or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been 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 **Exchange Data with External** time or the schedule is manually invoked, using the Monk function **start-schedule** (see **Chapter 5** for more information).

The default is **LNExchange.**

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 **LNConnect.**

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 **LNVerify.**

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.

The default is **LNShutdown.**

Any return value indicates that the **suspend** command can proceed and that the connection to the external system can be broken immediately.

Positive Acknowledgment Function

Description

Specifies a Monk function that the e*Way will call when *all* the Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined.

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 **LNAck.**

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 **LNNak.**

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, as follows:

- 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 40).

The default is **LNNotify.**

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. Lotus Notes Settings

The parameters in this section control access to a Lotus Notes database.

Note: *Make sure you enter the user name first then the password.*

Lotus Notes Server Name

Description

Specifies the name of the Lotus Notes Server. Use the server name, usually in the format of NotesBox/NotesCert, or Local.

Required Values

Any vaild string. The default is NotesBox/NotesCert.

Password for the Notes Server

Description

Specifies the user password that provides access to the Lotus Notes database.

Required Values

Any valid string.

3.2 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.*

3.3 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

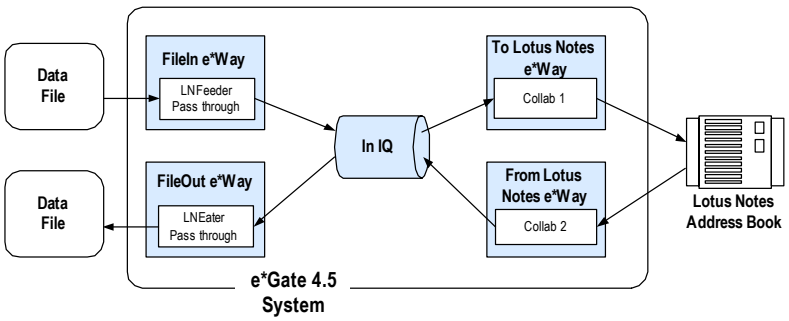
Implementation

This chapter provides instructions for installing the sample schema from the e*Gate CD.

4.1 Functional Overview of the Sample Schema

In this sample the Lotus Notes e*Way intercepts the input file (Event), applies associated Collaborations and Rules in relation to the Lotus Notes Address Book, and outputs the results to a specified location.

Figure 8 Sample schema basic architecture



The Lotus Notes Schema—Overview

1. The FileIn e*Way reads the inbound information from the external data file and publishes it as an Event to the IQ.
2. The ToLotusNotes e*Way subscribes to the inbound Event from the FileIn e*Way, transforms the Event, and publishes it to the Lotus Notes (address book) system.
3. The Lotus Notes system processes the request and returns the reply to the FromLotusNotes e*Way.
4. The FromLotusNotes e*Way receives the reply from the Lotus Notes (address book) system, transforms the Event, and publishes it to the IQ.
5. The FileOut e*Way subscribes to the outbound Event from the [Lotus Notes e*Way] and publishes it to the output data file.

4.2 Installing the Lotus Notes Sample Schema

The e*Gate Installation CD contains a sample schema to demonstrate a simple scenario using the Lotus Notes e*Way.

4.2.1. Install the Sample Schema on the Registry Host

The following procedures are performed on the host machine.

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

1. Start the e*Gate Schema Designer GUI.
2. When the Schema Designer prompts you to log in, select the host that you specified during installation, and enter your password.
3. You will then be prompted to select a schema. Click **New**.
4. Enter a name for the new Schema. In this case, enter **LotusNotesSample**, or any appropriate name as desired.
5. Select **Create from export**, locate the **LotusNotesSample.zip** on the CD and click **Open**. The repository files will be imported into the **LotusNotesSample** schema.

The e*Gate Schema Designer opens to your new schema. You are now ready to begin creating the necessary components for this sample schema.

4.2.2. Files Included with the Sample Schema

Importing the Lotus Notes e*Way sample schema will install files shown in Table 2 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 2 Files for the Sample Schema

e*Gate Client Directory	File(s)
monk_scripts\common\	AddressBook.ssc AddressBookExtract.dsc AddressBookInsert.tsc
data\LotusNotes\	LotusNotes.dat
configs\stcewgenericmonk\	FromLotusNotes.cfg FromLotusNotes.sc ToLotusNotes.cfg ToLotusNotes.sc

Table 2 Files for the Sample Schema (Continued)

e*Gate Client Directory	File(s)
configs\stcewfile\	LNeater.cfg LNeater.sc LNfeeder.cfg LNfeeder.sc

Chapter 5

Lotus Notes e*Way Functions

This chapter explains the functions used to control the Lotus Notes e*Way's basic operations, as well as the functions included within the library (file name **stc_monklnotes.dll**).

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

- **Basic Functions** on page 35
- **Lotus Notes Functions** on page 43

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

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 36*

get-logical-name on page 37

send-external-down on page 38

send-external-up on page 39

shutdown-request on page 40

start-schedule on page 41

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown request requests the e*Way to perform the shutdown procedure when there is no outstanding incoming/outgoing event. When the e*Way is ready to act on the shutdown request, in invokes the Shutdown Command Notification Function (see**"Shutdown Command Notification Function" on page 30**). Once this function is called, the shutdown proceeds immediately.

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

Parameters

None.

Return Values

None.

Throws

None.

start-schedule

Syntax

```
(start-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

5.2 Lotus Notes Functions

The current suite of Lotus Notes e*Way Monk functions support read-only activities.

Functions included within the library file **stc_monklnotes.dll** are

DBHandleOK on page 45
GetDBHandleStatus on page 46
GetFieldData on page 47
GetFieldList on page 48
GetNotHandleByUNID on page 49
LNAck on page 50
LNConnect on page 51
LNExchange on page 52
LNGetDBHandle on page 53
LNInit on page 54
LNNak on page 55
LNNotesRead on page 56
LNNotify on page 57
LNOutgoing on page 58
LNShutdown on page 59
LNStartup on page 60
LNVerify on page 61
MarkAsResponse on page 62
MarkNoteAsRead on page 63
MarkNoteAsUnRead on page 64
NextNoteHandle on page 65
NIFCloseCollection on page 66
NIFOpenCollection on page 67
NoteIDAsString on page 68
NoteHasParent on page 69
NotesInit on page 70
NotesTerm on page 71
NSFDbClose on page 72
NSFDbOpen on page 73
NSFDbOpenNet on page 74
NSFItemAppendTextList on page 75
NSFItemGetTextListEntries on page 76
NSFItemGetTextListEntry on page 77
NSFItemInfo on page 78
NSFItemSetNumber on page 79
NSFItemSetText on page 80
NSFNoteClose on page 81
NSFNoteCreate on page 82
NSFNoteDelete on page 83
NSFNoteUpdateExtended on page 84
OpenNoteByID on page 85
SendMail on page 86

General Usage Notes

Closing Handles

It is unnecessary to explicitly invoke **NSFClose**, **NIFCloseCollection** or **NSFNoteClose** to close any handle returned by any of these routines, because a handle is closed as it becomes unbound, and any resources associated with it are relinquished.

Monk Error Objects

The "Return Value" sections for each function list the expected results of each function. Keep in mind that incorrect usage and internal errors generate Monk error objects dealt with by the Monk engine.

Flag Strings

Various flags are available to the C API, which impose control over the behavior of some functions. At the C level, these flags are macros equating to numeric values. These values are unavailable at the Monk level. Therefore, the Monk developer must refer to these macro constants by their literal names.

Flags may be effectively OR'd, bit-wise by using the pipe "|" symbol, for example,

```
(NSFNoteUpdateExtended noteHandle "UPDATE_NOCOMMIT |
UPDATE_NOREVISION")
```

White space on either side of "|" is not necessary.

Knowledge of the C API and macro names is required in order to use them correctly. If no flags are required the developer may allow the function to pass a null string, as follows:

```
(NSFNoteUpdateExtended noteHandle "")
```

DBHandleOK

Syntax

```
(DBHandleOK handle)
```

Description

DBHandleOK determines the validity of a database handle returned from **NSFDbOpen** or **NSFDbOpenNet** functions.

Parameters

Name	Type	Description
handle	Handle	A database handle.

Return Values

Boolean

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

Throws

None.

Examples

```
(let ((hDB (NSFDbOpen "foo.nsf")))
   (if (DBHandleOK hDB)
       (begin
          ; We have a valid database handle
       )
       (begin
          (display "Could not open database.Error code=")
          (display (number->string (GetDBHandleStatus hDB)))
          (newline)
       )
   )
)
```

GetDBHandleStatus

Syntax

```
(GetDBHandleStatus handle)
```

Description

GetDBHandleStatus returns the internal status code from a database handle. The code number is Lotus C API-specific but could be useful in determining specific error conditions related to the opening of the database.

Parameters

Name	Type	Description
handle	Handle	A database handle.

Return Values

integer
Returns a positive integer.

Note: *A return value of zero does not imply that the handle being tested is available for use in other procedure calls. Specifically, if the handle has been closed following a successful invocation of* ***NSFDbOpen*** *or* ***NSFDbOpenNet****, this procedure returns zero (see* **NSFDbOpen** *on page 73 and* **NSFDbOpenNet** *on page 74). Use* ***DBHandleOK*** *for stronger validation (see* **DBHandleOK** *on page 45).*

Throws

None.

Examples

```
(let ((hDB (NSFDbOpen "foo.nsf")))
    (if (DBHandleOK hDB)
        (begin
        ; We have a valid database handle
        )
        (begin
        (display "Could not open database. Error code=")
        (display (number->string (GetDBHandleStatus hDB)))
        (newline)
        )
    )
)
```

GetFieldData

Syntax

```
(GetFieldData handle fieldname)
```

Description

GetField Data acquires the data associated with a specified field within a Note.

Parameters

Name	Type	Description
handle	Handle	A note handle.
fieldname	String	A name of a field.

Return Values

string
Returns a string object, with a possible length of zero.

Throws

None.

Examples

```
(let* ((field "name")(value (GetFieldData hNote field)))
    (if value
        (display (string-append field "=" value "\n"))
        (display (string-append field " does not exist\n"))
    )
)
```

GetFieldList

Syntax

```
(GetFieldList handle)
```

Description

GetFieldList determines the names of all the fields in a Note.

Parameters

Name	Type	Description
handle	Handle	A valid note handle.

Return Values

string
Returns a Monk list of strings.

Throws

None.

Examples

```
; display the names of all fields in a Note
    (let ((masterList (GetFieldList hNote)))
        (if (list? masterList)
            (do ((list masterList (cdr list)))
                ((null? List) #t)
                (display (car list))
                (newline)
            )
        )
    )
```

GetNotHandleByUNID

Syntax

```
(GetNoteHandleByUNID <dbH> <note_UNID> )
```

Description

GetNotHandleByUNID gets a note handle through its unique ID. If note_UNID is a null string then a new note is created.

Parameters

Name	Type	Description
dbh	Handle	A valid database handle.
note_UNID	String	A Monk string previously returned from NoteIDAsString.

Return Values

Handle or Boolean

A valid Lotus Notes handle or the boolean #f (false) upon failure.

Throws

None.

LNAck

Syntax

```
(LNAck arg)
```

Description

LNAck is a dummy ACK function.

Parameters

Name	Type	Description
arg	Any	Parameter not used at this time.

Return Values

String

Empty string.

Throws

None.

LNConnect

Syntax

```
(LNConnect )
```

Description

LNConnect initalizes connection to the Lotus Notes system.

Parameters

None

Return Values

String

Always returns the string “UP” indicates connection is up (operational).

Throws

None.

LNExchange

Syntax

```
(LNExchange)
```

Description

LNExchange is a dummy Exchange Data function.

Parameters

None

Return Values

String
Empty string.

Throws

None.

LNGetDBHandle

Syntax

```
(LNGetDBHandle <ln-adm-node> )
```

Description

LNGetDBHandle returns a Lotus Notes database handle based on the Lotus Notes ETD structure.

Parameters

Name	Type	Description
ln-adm-node	Path	Path to the ADM node which contains the name and view.

Return Values

Handle or Boolean

A valid Lotus Notes database handle, or a boolean #f (false) upon failure.

Throws

None.

LNInit

Syntax

```
(LNInit)
```

Description

LNInit does global initializations for Lotus Notes eWay.

Parameters

None

Return Values

String
An empty string.

Throws

None.

LNNak

Syntax

```
(LNNak arg)
```

Description

LNNak is a dummy NAK function

Parameters

Name	Type	Description
arg	Any	Parameter not used at this time.

Return Values

String

Empty string.

Throws

None.

LNNotesRead

Syntax

```
(LNNotesRead <dbV> <dbC> <allNotes> <ln-root-node>)
```

Description

LNNotesRead allows the user to obtain a collection of Lotus Notes based on the Lotus Notes ETD structure.

Parameters

Name	Type	Description
dbV	Handle	The location of the database where the View is found.
dbC	Handle	The database where the Collection will be formed.
allNotes	boolean	If set to "true" (#t) all Notes from the Collection will be returned. If set to "false" (#f) only unread Notes will be returned from the Collection.
ln-root-node	Path	Root node of the ETD that will populated with data returned from Lotus Notes.

Return Values

Integer or Boolean

The number of Lotus Notes obtained or the boolean #f (false) upon failure.

Throws

None.

LNNotify

Syntax

```
(LNNotify command)
```

Description

LNNotify is a dummy notify function

Parameters

Name	Type	Description
command	Any	Parameter not used at this time.

Return Values

String
Empty string.

Throws

None.

LNOutgoing

Syntax

```
(LNOutgoing msg)
```

Description

LNOutgoing is a dummy outgoing function

Parameters

Name	Type	Description
msg	Any	Parameter not used at this time

Return Values

String

Empty string.

Throws

None.

LNShutdown

Syntax

```
(LNShutdown command)
```

Description

LNShutdown is a dummy shutdown function.

Parameters

Name	Type	Description
command	Any	Parameter not used at this time

Return Values

String
Empty string.

Throws

None.

LNStartup

Syntax

```
(LNStartup)
```

Description

LNStartup initializes connection with Lotus Notes.

Parameters

None

Return Values

String
Returns "UP" upons success, empty string upon failure.

Throws

None.

LNVerify

Syntax

```
(LNVerify)
```

Description

LNVerify verifies connection to Lotus Notes.

Parameters

None

Return Values

String
Always returns the string “UP” indicates connection is up (operational).

Throws

None.

MarkAsResponse

Syntax

```
(MarkAsResponse parentNote childNote)
```

Description

MarkAsResponse identifies a Note as being a response to another Note.

Parameters

Name	Type	Description
parentNote	String	A type of Note.
childNote	String	A response to a parentNote.

Return Values

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

Throws

None.

Examples

```
(MarkAsResponse hNoteP hNoteC)
```

MarkNoteAsRead

Syntax

```
(MarkNoteAsRead handleD handleN)
```

Description

MarkNoteAsRead marks the specified Note as having been read.

Parameters

Name	Type	Description
handleD	Handle	A valid database Handle.
handleN	Handle	A note handle.

Return Values

Boolean
Returns **#t** (true) if the respective handles are valid; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(MarkNoteAsRead dbHandle noteHandle)
```

MarkNoteAsUnRead

Syntax

```
(MarkNoteAsUnRead handleD handleN)
```

Description

MarkNoteAsUnRead marks the specified Note as unread.

Parameters

Name	Type	Description
handleD	Handle	A valid database handle.
handleN	Handle	A note handle.

Return Values

Boolean
Returns **#t** (true) if the respective handles are valid; otherwise, returns **#f** (false).

Throws

None.

Examples

```
(MarkNoteAsUnRead dbHandle noteHandle)
```

NextNoteHandle

Syntax

```
(NextNoteHandle handleD handleC)
```

or

```
(NextNoteHandle handleD handleC boolean)
```

Description

NextNoteHandle enumerates a Collection (via its handle). The caller must specify whether all Notes or only unread Notes are returned.

Parameters

Name	Type	Description
handleD	Handle	The database handle where the Notes are found.
handleC	Handle	A collection handle from **NIFOpenCollection** function.
optional parameter	Boolean	A parameter used to determine whether unread Notes or all Notes are returned from the Collection. If set to "true" (**#t**) only unread Notes are to be returned from the Collection. If this parameter is "false" (**#f**) or missing, all Notes from the Collection will be returned.

Return Values

Returns a valid Note handle if successful; otherwise, returns Boolean **#f** (false).

Throws

None.

Examples

```
(NextNoteHandle dbHandle coHandle)
```

Note: *See the example on* **Examples** *on page 48.*

NIFCloseCollection

Syntax

```
(NIFCloseCollection handle)
```

Description

NIFCloseCollection closes a Collection handle and releases all associated resources.

Parameters

Name	Type	Description
handle	Handle	A valid return value from **NIFOpen** Collection function (see **NIFOpenCollection** on page 67).

Return Values

Boolean
Returns **#t** (true) always.

Throws

None.

Examples

```
(NIFCloseCollection coHandle)
```

NIFOpenCollection

Syntax

```
(NIFOpenCollection handleV handleC name)
```

Description

NIFOpenCollection opens a Notes Collection, using a View.

Parameters

Name	Type	Description
handleV	Handle	The location of the database where the View is found.
handleC	Handle	The database where the Collection will be formed.
name	String	The name of the View to be used.

Note: *The handles must be acquired with* ***NSFDbOpen*** *(see* **NSFDbOpen** *on page 73) or* ***NSFDbOpenNet*** *function (see* **NSFDbOpenNet** *on page 74).*

Return Values

Returns a valid Collection handle, or a Boolean **#f** (false) upon failure.

Throws

None.

Examples

```
(let ((coHandle (NIFOpenCollection dbHandle dbHandle "myview")))
   (if coHandle
     (begin
     ; we have a valid collection handle
     )
     (begin
     ; an error has occurred - read the error log
     )
   )
)
```

NoteIDAsString

Syntax

```
(NoteIDAsString handle bool)
```

Description

NoteIDAsString gets the unique ID (UNID) of a Note and/or the UNID of a Note's parent, both as a Monk string.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
bool	Boolean	An optional Monk Boolean object.

Return Values

A Monk string representing the required UNID in the following form:

"FXXXXXXXX:XXXXXXXX-NXXXXXXXX:XXXXXXXX"

Where XXXXXXXX are hexadecimal numbers.

If the second argument is present and is true (**#t**), the given Note's parent UNID will be returned. If there is no parent, the returned value is

"F00000000:00000000-N00000000:00000000"

This value does not represent any existing Note's UNID.

Consider using this option in conjunction with **NoteHasParent** (see **NoteHasParent** on page 69).

Throws

None.

Examples

```
(NoteIDAsString hNote #t)
```

or

```
(NoteIDAsString hNote #f)
```

These examples are equivalent to

```
(NoteIDAsString hNote)
```

NoteHasParent

Syntax

```
(NoteHasParent handle)
```

Description

NoteHasParent determines if the specified Note has a parent, that is, if the specified Note is a response Note.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.

Return Values

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

Throws

None.

Examples

```
(if (NoteHasParent hNote)
        (display "This is a response Note\n")
        (display "This Note is not a response Note\n")
)
```

NotesInit

Syntax

```
(NotesInit password)
```

Description

NotesInit establishes an environment allowing use of the Lotus Notes C API. All other functions will fail if this is not called or if it fails.

Parameters

Name	Type	Description
password	string	The user's password.

Return Value

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

Throws

None.

Example

```
(if (NotesInit "foo")
    (begin
    ; Initialization was successful
    )
    (display "Notes failed to initialize")
)
```

NotesTerm

Syntax

```
(NotesTerm)
```

Description

NotesTerm terminates the Notes session.

Parameters

None.

Return Values

None.

Throws

None.

Examples

```
(NotesTerm)
```

Additional Information

It is unnecessary to call this function explicitly because it is invoked automatically when (under Windows) the ***.dll** file is unloaded.

NSFDbClose

Syntax

```
(NSFDbClose handle)
```

Description

NSFDbClose closes a database handle.

Parameters

Name	Type	Description
handle	Handle	A valid handle acquired from either **NSFDbOpen** (see **NSFDbOpen** on page 73) or **NSFDbOpenNet** (see **NSFDbOpenNet** on page 74).

Return Values

Boolean

Returns **#t** (true) always.

Throws

None.

Examples

```
(let ((dbHandle (NSFDbOpen "foo.nsf"))
    (if dbHandle
    (NSFDbClose dbHandle)
    )
)
```

NSFDbOpen

Syntax

```
(NSFDbOpen databasename)
```

Description

NSFDbOpen opens a local database and returns a handle to it.

Parameters

Name	Type	Description
databasename	String	The name of the database to be opened.

Return Values

handle

Returns a database handle.

Note: *A returned database name is not necessarily valid. You may validate the object using* ***DBHandleOK*** *(see* **DBHandleOK** *on page 45**).*

Throws

None.

Examples

```
(let ((dbHandle (NSFDbOpen "Api462re.nsf")))
    (if (DBHandleOK dbHandle)
        (begin
           ; We have a valid database handle
        )
        (begin
         (display "Failed to open database\nError code=")
         (display (number->string (GetDBHandleStatus dbHandle)))
         (newline)
        )
    )
)
```

NSFDbOpenNet

Syntax

```
(NSFDbOpenNet databasename servername portname)
```

Description

NSFDbOpenNet opens a remote database and returns a handle to it.

Parameters

Name	Type	Description
databasename	String	The name of the database to be opened.
servername	String	The name of the Lotus Notes server.
portname	String	Optional Lotus Notes port name.

Return Values

handle

Returns a database handle, if successful.

Note: *A returned database handle is not necessarily valid. You may validate the object, using **DBHandleOK** (see* **DBHandleOK** *on page 45**).*

Examples

```
(let ((dbH (NSFDbOpenNet "Api462re.nsf" "joe.uk.xyzcorp.com/STC")))
    (if (DBHandleOK dbH)
        (begin
        ; We have a valid database handle
        )
        (begin
        (display "Failed to open database\nError code=")
        (display (number->string (GetDBHandleStatus dbH)))
        (newline)
        )
    )
)
```

NSFItemAppendTextList

Syntax

```
(NSFItemAppendTextList handle name text bool)
```

Description

NSFItemAppendTextList creates or appends to a text list item.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
name	String	A text list field item name.
text	String	A string of text.
bool	String	A string that indicates whether or not duplicates of the field item name are allowed.

Return Values

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

Examples

```
(NSFItemAppendTextList hNote "Categories" "A" #f)
```

Additional Information

If the field name specified in the second parameter (*name*) does not exist, it is created and the last parameter is not required.

If the *text* being passed contains a semicolon, ambiguous values could be returned from **GetFieldData** (see **GetFieldData** on page 47). This is because the Notes' internal mechanism for presenting a text list item as a single value is to add a semicolon separator to the individual elements. SeeBeyond recommends that you use **NSFItemGetTextListEntry** (see **NSFItemGetTextListEntry** on page 77) for this type of data. If data with a semicolon is added to a text list, a warning is written to the log.

If the last parameter (*bool*) is **#f**, and an attempt is made to append a duplicate value to the text list, the procedure returns true (**#t**). This return is not an error.

NSFItemGetTextListEntries

Syntax

```
(NSFItemGetTextListEntries handle name)
```

Description

NSFItemGetTextListEntries determines the number of entries (elements) in a text list item.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
name	String	A field name.

Return Values

integer

Returns the number of entries in the specified field (as a Monk number object, integer exact).

Throws

None.

Examples

```
(NSFItemGetTextListEntires hNote "Categories")
```

NSFItemGetTextListEntry

Syntax

```
(NSFItemGetTextListEntry handle name element)
```

Description

NSFItemGetTextListEntry retrieves the "Nth" element in a text list field item.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
name	String	A field name.
element	Integer	A base-zero element number expressed as an integer.

Return Values

Returns a Monk string object containing the requested data; otherwise, Boolean **#f** (false) in case of an error.

Throws

None.

Examples

```
; dump the contents of each element of the "Category" field item
(assumed to be a text list item)
(let* ((field "Categories")(elements (NSFItemGetTextListEntries hNote
field)))
     (if (number? elements)
         (do ((i 0 (+ i 1)))
         ((>= i elements) #t)
         (display (NSFItemGetTextListEntry hNote field i))
         (newline)
         )
     )
)
```

NSFItemInfo

Syntax

```
(NSFItemInfo handle item)
```

Description

NSFItemInfo determines the type of item (field) within a Note.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
item	String	The name of a field expected to be found in the Note.

Return Values

string

Returns a string object of the form TYPE_xxx, as follows:

TYPE_ACTION	TYPE_QUERY
TYPE_ASSISTANT_INFO	TYPE_SCHED_LIST
TYPE_CALENDAR_FORMAT	TYPE_SEAL
TYPE_COLLATION	TYPE_SEAL_LIST
TYPE_COMPOSITE	TYPE_SEALDATA
TYPE_ERROR	TYPE_SIGNATURE
TYPE_FORMULA	TYPE_TEXT
TYPE_HIGHLIGHTS	TYPE_TEXT_LIST
TYPE_HTML	TYPE_TIME
TYPE_ICON	TYPE_TIME_RANGE
TYPE_INVALID_OR_UNKNOWN	TYPE_UNAVAILABLE
TYPE_LSOBJECT	TYPE_USERDATA
TYPE_NOTELINK_LIST	TYPE_USERID
TYPE_NOTEREF_LIST	TYPE_VIEW_FORMAT
TYPE_NUMBER	TYPE_VIEWMAP_DATASET
TYPE_NUMBER_RANGE	TYPE_VIEWMAP_LAYOUT
TYPE_OBJECT	TYPE_WORKSHEET_DATA

Throws

None.

Examples

```
(display (string-append "Field 'Age' is of type "
 (NSFItemInfo hNote "Age") "\n"))
```

NSFItemSetNumber

Syntax

```
(NSFItemSetNumber handle name value)
```

Description

NSFItemSetNumber adds/updates an item (field) and its numeric value in a Note.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
name	String	The item (field) name.
value	String, integer or real value	A numeric value represented in one of three ways.

Return Values

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

Throws

None.

Examples

```
(NSFItemSetNumber noteHandle "Age"  "27")
```

or

```
(NSFItemSetNumber noteHandle "Age"  27)
```

or

```
(NSFItemSetNumber noteHandle "Age"  27.5)
```

NSFItemSetText

Syntax

```
(NSFItemSetText handle name value)
```

Description

NSFItemSetText adds/updates an item (field) and its text value in a Note.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
name	String	The item (field) name.
value	String	The text string to be associated with the item.

Return Values

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

Throws

None.

Examples

```
(NSFItemSetText noteHandle "Name"  "Sam")
```

NSFNoteClose

Syntax

```
(NSFNoteClose handle)
```

Description

NSFNoteClose closes a Note.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle from **NextNoteHandle** (see **NextNoteHandle** on page 65).

Return Values

Boolean

Returns **#t** (true) always.

Throws

None.

Examples

```
(NSFNoteClose noHandle)
```

NSFNoteCreate

Syntax

```
(NSFCreateNote handle)
```

Description

NSFNoteCreate creates a new Note in memory.

Parameters

Name	Type	Description
handle	Handle	A valid database handle.

Return Values

handle
Returns a Note handle if successful; returns a Boolean **#f** (false) if an error occurs.

Throws

None.

Examples

```
(let (( noHandle (NSFNoteCreate dbHandle)))
    (if noHandle
        (begin
            ; We have an in-memory Note
        )
        (begin
            ; Check the error log - something horrible has
happened
        )
    )
)
```

Note: *The initial creation of a Note gives a handle on an in-memory image. The Note will not exist in the database unless and until* ***NSFNoteUpdateExtended*** *has been called successfully (see* **NSFNoteUpdateExtended** *on page 84**).*

NSFNoteDelete

Syntax

```
(NSFNoteDelete handleD handleN flags)
```

Description

NSFNoteDelete deletes a Note.

Parameters

Name	Type	Description
handleD	Handle	A database handle where Notes are found.
handleN	Handle	A Note handle.
flags	String	A flag string (see **Flag Strings** on page 44).

Return Values

Boolean

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

Throws

None.

Examples

```
(NSFNoteDelete dbHandle noteHandle "")
```

NSFNoteUpdateExtended

Syntax

```
(NSFNoteUpdateExtended handle flags)
```

Description

NSFNoteUpdateExtended updates (writes to database) an in-memory Note.

Parameters

Name	Type	Description
handle	Handle	A valid Note handle.
flags	String	A flag string (see **Flag Strings** on page 44).

Return Values

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

Throws

None.

Examples

```
(NSFNoteUpdateExtended noteHandle "UPDATE_NOCOMMIT")
```

OpenNoteByID

Syntax

```
(OpenNoteByID handle unid)
```

Description

OpenNoteByID opens a Note given a handle on the database containing the Note and its UNID.

Parameters

Name	Type	Description
handle	Handle	A valid database handle.
unid	String	A Monk string previously returned from **NoteIDAsString** (see NoteIDAsString on page 68).

Return Values

Returns a Note handle; returns a Boolean **#f** (false) if the UNID does not match a Note in the specified database.

Throws

None.

Examples

```
(let ((hNote (OpenNoteByID handle unid)))
    (if hNote
        (display "Got Note\n")
        (display "Failed to get Note\n")
    )
)
```

SendMail

Syntax

```
(SendMail sendTo copyTo subject body)
```

Description

SendMail creates and sends an E-mail item.

Parameters

Name	Type	Description
sendTo	String	An e-mail address or list of addresses.
copyTo	String	An e-mail address or list of addresses.
subject	String	The e-mail subject line.
body	String	The body (text) of the e-mail.

Return Values

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

Throws

None.

Example No. 1

```
(SendMail "andy@xyzcorp.com;fred@xyzcorp.com"
"john@xyzcorp.com;gail@xyzcorp.com"
 "Just testing" "Dear Everyone:\nThis is just a test.\nBest
wishes,\nJoe")
```

Additional Information

Note the following properties of the previous example:

- Address separators may be a semicolon (;), colon (:), or comma (,).
- The body of the text must be passed as a Monk string containing newline characters to delimit each line of text. The delimiter is not necessary for the last line.

Example No. 2 on page 87 loads the ***.dll** file and initializes the Lotus Notes run-time environment. The file then opens the (local) database **Api462re.nsf**.

In this example the View is in the same database as the Notes data to be collected. The View name is **($All)**. All Note handles that constitute this Collection are enumerated. For each Note, the values associated with the fields **Name** and **ShortDesc** are then displayed.

The handle closing routines have been commented out because they are not necessary unless early resource release is required.

Example No. 2

```
(load-extension "d:/Notes/notesDLL/release/NotesDLL.dll")

(define myPassword "abc123") ; Specify the password

(if (NotesInit myPassword) ; Initialise Notes session
    (let ((dbH (NSFDbOpen "d:/notesapi/doc/Api462re.nsf")))
        (if (DBHandleOK dbH)
            (let ((dbC (NIFOpenCollection dbH dbH "All \
Alphabetical")))
                (if dbC
                    (do ((dbN (NextNoteHandle dbH dbC)
(NextNoteHandle dbH dbC)))
                        ((not dbN) #t)
                        (display (GetFieldData dbN "Name"))
                        (display ", ")
                        (display (GetFieldData dbN "ShortDesc"))
                        (newline)
                        ; (NSFNoteClose dbN)
                    )
                    ; (NIFCloseCollection dbC)
                )
                ; (NSFDbClose dbH)
            )
        )

        (NotesTerm) ; Terminate the session
    )
    (display "Notes initialisation failed\n")
)
```

Index

A

Additional Path parameter **24**
Auxiliary Library Directories parameter **24**

B

book **32**

C

configuration parameters **11**
  Additional Path **24**
  Auxiliary Library Directories **24**
  changing, how to **11**
  Down Timeout **14**
  Exchange Data Interval **14**
  Exchange Data With External Function **26**
  External Connection Establishment Function **27**
  External Connection Shutdown Function **28**
  External Connection Verification Function **28**
  Forward External Errors **12**
  General Settings **11**
  Journal File Name **12**
  Lotus Notes settings **30**
    Password **30**
    Server Name **30**
  Max Failed Messages **12**
  Max Resends Per Message **12**
  Monk Environment Initialization File **25**
  Negative Acknowledgment Function **29**
  Positive Acknowledgement Function **28**
  Process Outgoing Message Function **26**
  Resend Timeout **15**
  Shutdown Command Notification Function **30**
  Start Exchange Data Schedule **14**
  Startup Function **25**
  Stop Exchange Data Schedule **14**
  Up Timeout **14**
  Zero Wait Between Successful Exchanges **15**

D

DBHandleOK **45**
Down Timeout parameter **14**

E

e*Way components **6**
  stcewgenericmonk.exe 6
e-mail **6, 86**
Exchange Data Interval parameter **14**
Exchange Data with External Function parameter **26**
External Connection Establishment Function parameter **27**
External Connection Shutdown Function parameter **28**
External Connection Verification Function parameter **28**
external systems requirements **7**

F

Files and Directories Created **9**
flag strings **44**
Forward External Errors parameter **12**
functions, Lotus Notes e*Way **35**
  basic **35**
    event-send-to-egate **36**
    get-logical-name **37**
    send-external-down **38**
    start-schedule **41**
    stop-schedule **42**
  Lotus Notes specific
    43
  specific
    DBHandleOK **45**
    GetDBHandleStatus **46**
    GetFieldData **47**
    GetFieldList **48**
    MarkAsResponse **62**
    MarkNoteAsRead **63**
    MarkNoteAsUnRead **64**
    NextNoteHandle **65**
    NIFCloseCollection **66**
    NIFOpenCollection **67**
    NoteHasParent **69**
    NoteIDAsString **68**
    NotesINit **70**
    NotesTerm **71**
    NSFDbClose **72**
    NSFDbOpen **73**
    NSFDbOpenNet **74**
    NSFItemAppendTextList **75**
    NSFItemGetTextListEntries **76**
    NSFItemGetTextListEntry **77**
    NSFItemInfo **78**
    NSFItemSetNumber **79**
    NSFItemSetText **80**
    NSFNoteClose **81**

NSFNoteCreate **82**
NSFNoteDelete **83**
NSFNoteUpdateExtended **84**
OpenNoteByID **85**
SendMail **86**

G

GetDBHandleStatus **46**
GetFieldData **47**
GetFieldList **48**

I

implementation **32**
sample schema, overview **32**
index
book **32**
installation
Lotus Notes sample schema **33**
Installation Procedure
Windows **8**
intended reader **6**

J

Journal File Name parameter **12**

L

Lotus Notes Client **8**
Lotus Notes Server **8**

M

MarkAsResponse **62**
MarkNoteAsRead **63**
MarkNoteAsUnRead **64**
Max Failed Messages parameter **12**
Max Resends Per Message parameter **12**
Monk Environment Initialization File parameter **25**

N

Negative Acknowledgment Function parameter **29**
NextNoteHandle **65**
NIFCloseCollection **66**
NIFOpenCollection **67**
NoteHasParent **69**
NoteIDAsString **68**
NotesInit **70**
NotesTerm **71**
NSFDbClose **72**
NSFDbOpen **73**
NSFDbOpenNet **74**
NSFItemAppendTextList **75**
NSFItemGetTextListEntries **76**
NSFItemGetTextListEntry **77**
NSFItemInfo **78**
NSFItemSetNumber **79**
NSFItemSetText **80**
NSFNoteClose **81**
NSFNoteCreate **82**
NSFNoteDelete **83**
NSFNoteUpdateExtended **84**

O

OpenNoteByID **85**

P

Password **8**
Positive Acknowledgment Function parameter **28**
Process Outgoing Message Function parameter **26**

R

Resend Timeout parameter **15**

S

SendMail **86**
Shutdown Command Notification Function parameter **30**
Start Exchange Data Schedule parameter **14**
start-schedule **41**
Startup Function parameter **25**
Stop Exchange Data Schedule parameter **14**
stop-schedule **42**
supported operating systems **7**

U

Up Timeout parameter **14**

W

Windows
installation **8**

Z

Zero Wait Between Successful Exchanges **15**