SeeBeyond™ eBusiness Integration Suite

e*Way Intelligent Adapter for COM/DCOM User's Guide

Release 4.5.4

Monk Version

SᴇᴇBᴇʏᴏɴᴅ™

Contents

Introduction

The e*Way Intelligent Adapter for COM/DCOM enables the e*Gate system to exchange data with the server side of COM/DCOM-enabled applications. This document describes how to install, configure and implement the COM/DCOM e*Way.

This Chapter Explains:

- **"Intended Reader" on page 6**
- **"Components" on page 6**
- **"Supported Operating Systems" on page 6**
- **"System Requirements" on page 7**

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 or Windows 2000 operations and administration; to be thoroughly familiar with COM/DCOM; and to be thoroughly familiar with Windows-style GUI operations.

1.2 Components

The COM/DCOM 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, discussed in **"COM/DCOM e*Way Functions" on page 51**.

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

1.3 Supported Operating Systems

The COM/DCOM e*Way is available on the following operating systems:

- Windows 2000, Windows 2000 SP1, Windows 2000 SP2, and Windows 2000 SP3

- Windows NT 4 SP6a

1.4 System Requirements

To use the COM/DCOM e*Way, you need the following:

- An e*Gate Participating Host, version 4.5.1 or 4.5.2.

- 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.

1.5 External System Requirements

- The COM/DCOM e*Way requires an existing automation compatible COM/DCOM-compliant application or component.

Note: *One of the standard interfaces exposed by many applications and user-created objects is the IDispatch or "automation" interface. Windows NT and Windows 2000 expose many system services via the automation interface as do several of the Microsoft Office products like Excel. Additionally, the user can also create their own objects using programming languages like C, C++, java, Visual Basic, COBOL, etc. The COM/DCOM e*Way provides access to these objects using the IDispatch interface and semantics. The e*Way can dynamically create an instance of one of these components and then call methods on it.*

Installation

This chapter contains information on pre-installation requirements and the procedures for installing the COM/DCOM e*Way. In addition, a list of installed files is provided, along with the directories these files are located in.

This Chapter Explains:

2.1 Windows NT and Windows 2000 Installation

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.

2.1.1. Installation Procedure

To install the COM/DCOM e*Way on Windows NT or Windows 2000 systems:

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

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

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

4 The InstallShield setup application 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 Enterprise Manager.

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

7   Display the new e*Way's properties.

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

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

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

11 From the **e*Way Template Selection** list, select **stcewmscom** e*Way and click **OK**.

Figure 1   e*Way Template Selection
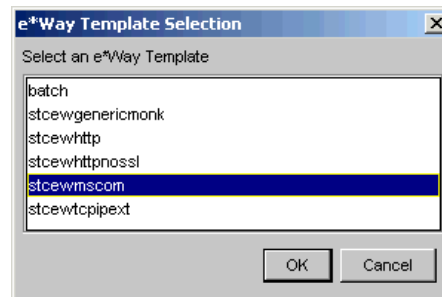
12 The e*Way Editor launches. Make any necessary changes, then save the configuration file.

13 You return to the e*Way's property sheet. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 4**.

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

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

2.2 Files/Directories Created by the Installation

The COM/DCOM e*Way installation process installs the following files within the e*Gate "client" directory tree. Files are 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)
eGate\Server\registry\repository\default\bin\	stcewgenericmonk.exe
configs\stcewgenericmonk\	stcewmscom.def
client\bin\ server\registry\repository\default\bin\win32	stc_monkcom.dll stccombuilder.exe
monk_library\	monkcom.gui
monk_library\ewmscom\	com-startup.monk com-struct-call.monk

Functional Overview

This chapter describes the basic operation of a typical e*Way based on the Generic e*Way Kernel. Additionally, basic information regarding the functionality of COM/DCOM is given.

This Chapter Explains:

- **"Typical e*Way Architecture" on page 11**
- **"COM/DCOM Basics" on page 19**

3.1 Typical e*Way Architecture

Architecturally, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers that handle communication with the external application, built upon an e*Way Kernel layer that manages the processing of data and subscribing or publishing to other e*Gate components (see Figure 2).

Figure 2   Typical e*Way Architecture
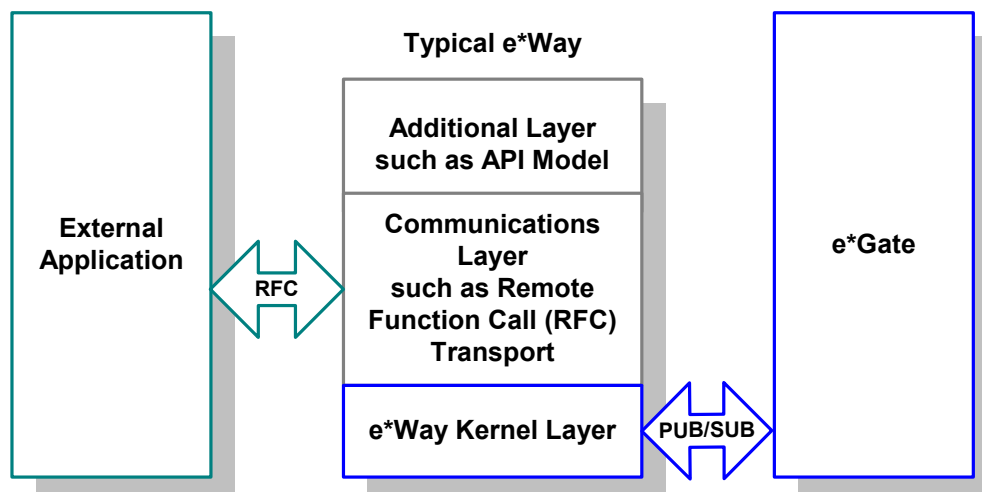
Each layer contains Monk scripts and/or functions, and makes use of lower-level Monk functions residing in the layer beneath. You, as user, primarily use the highest-level functions, which reside in the upper layer(s).

The upper layers of the e*Way use Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events," send those Events to Collaborations, and manage the connection between the e*Way and the external system (see Figure 3).

Figure 3 Basic e*Way Operations
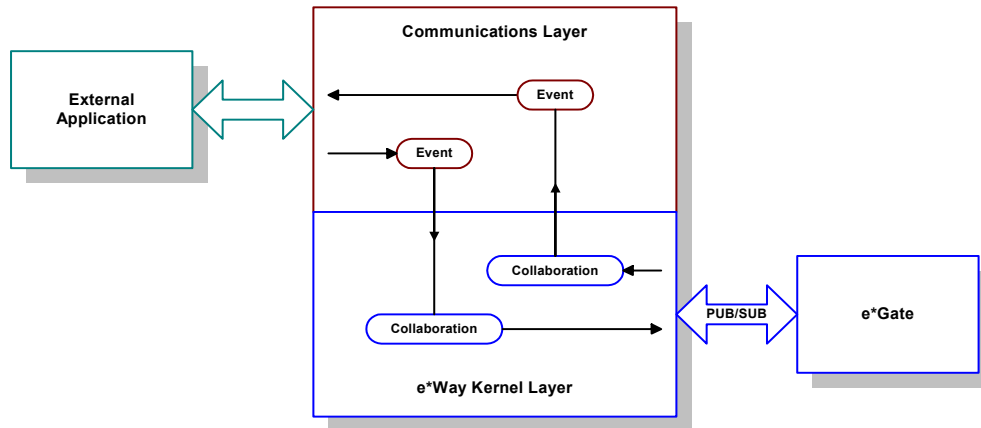
Configuration options that control the Monk environment and define the Monk functions used to perform these basic e*Way operations are discussed in **Chapter 4**. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft WordPad* or *Notepad*).

The upper layers of the e*Way are single-threaded. Functions run serially, and only one function can be executed at a time. The e*Way Kernel 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.

The basic set of e*Way Kernel Monk functions is described in **Chapter 6**. Generally, e*Way Kernel Monk functions should be called directly only when there is a specific need not addressed by higher-level Monk functions, and should be used only by experienced developers.

Basic e*Way Processes

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

Type of Operation	Name
Initialization	**"Startup Function" on page 33** (also see **"Monk Environment Initialization File" on page 32**)
Connection	**"External Connection Establishment Function" on page 35** **"External Connection Verification Function" on page 35** **"External Connection Shutdown Function" on page 36**

Type of Operation	Name
Schedule-driven data exchange	**"Exchange Data with External Function" on page 34** **"Positive Acknowledgment Function" on page 36** **"Negative Acknowledgment Function" on page 37**
Shutdown	**"Shutdown Command Notification Function" on page 37**
Event-driven data exchange	**"Process Outgoing Message Function" on page 33**

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

Initialization Functions

Figure 4 illustrates how the e*Way executes its initialization functions.

Figure 4   Initialization Functions
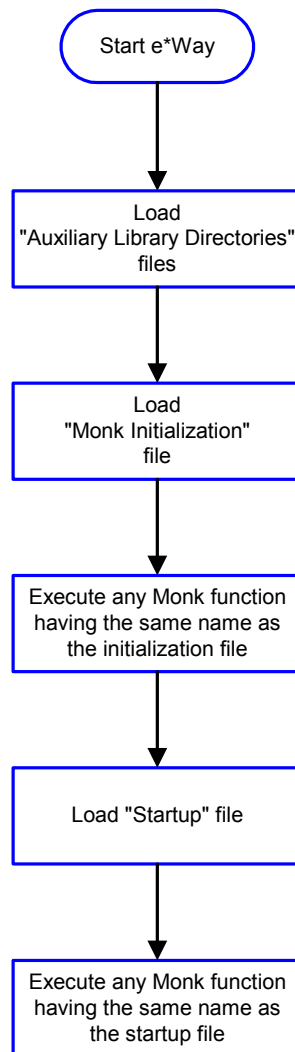
Connection Functions

Figure 5 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 5   Connection Establishment and Verification Functions
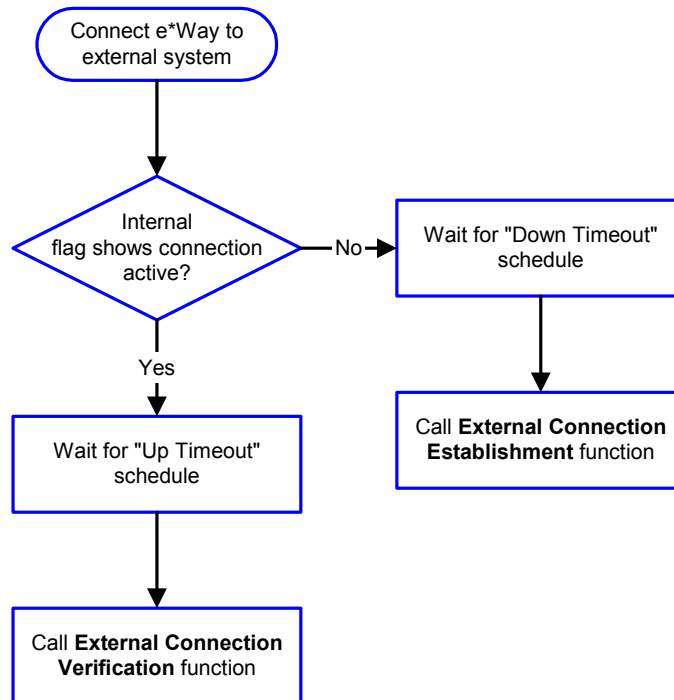
Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 7 on page 16** *and* **Figure 9 on page 18** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 55 and* **send-external-down** *on page 54 for more information.*

Figure 6 illustrates how the e*Way executes its "connection shutdown" function.

Figure 6   Connection Shutdown Function
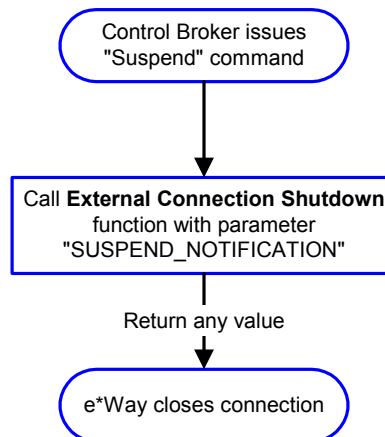
Schedule-driven Data Exchange Functions

Figure 7 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

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

- The "Start Data Exchange" time occurs.

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

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

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

Figure 7   Schedule-driven Data Exchange Functions
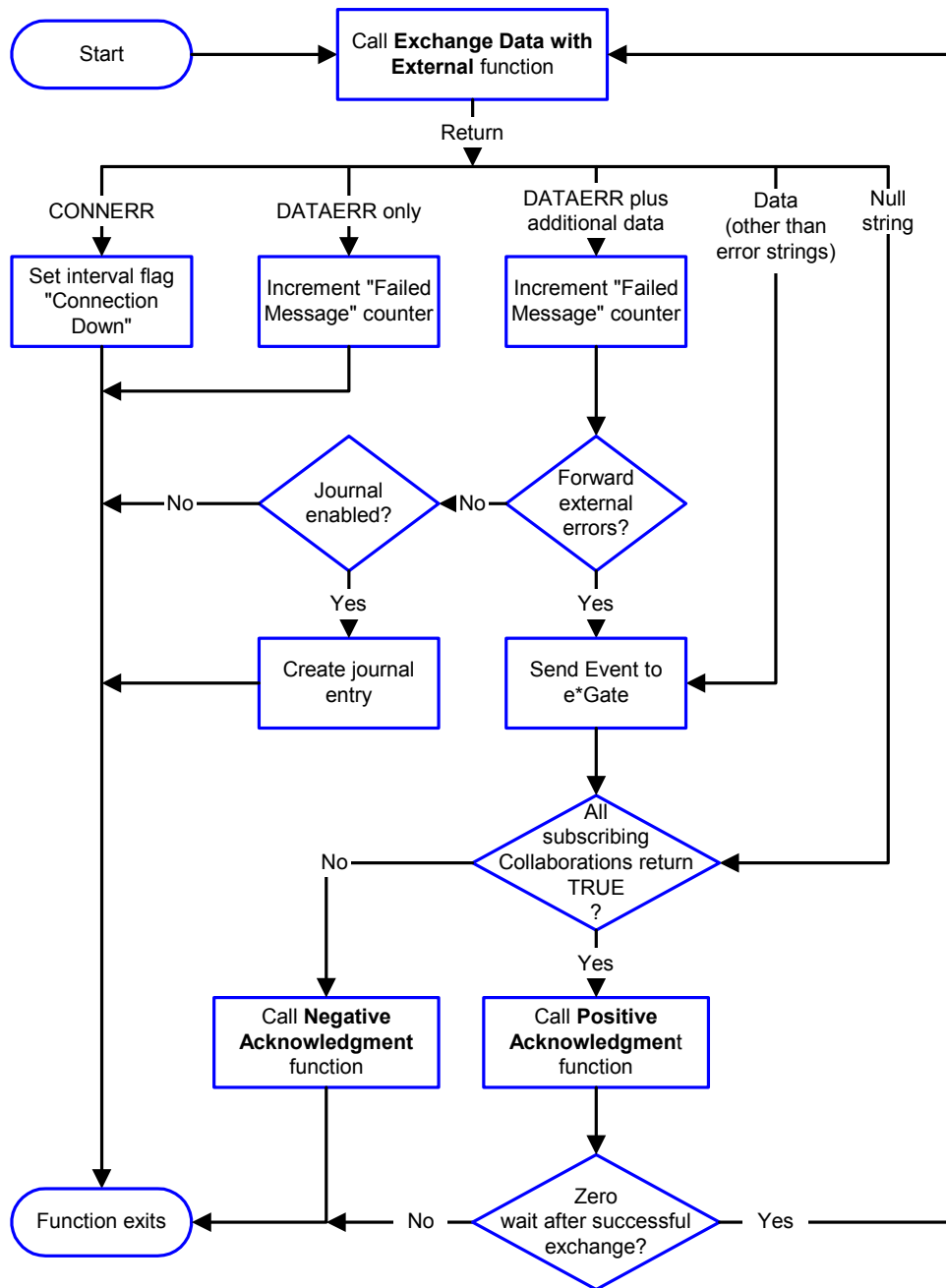
Shutdown Functions

Figure 8 illustrates how the e*Way implements the shutdown request function.

Figure 8 Shutdown Functions
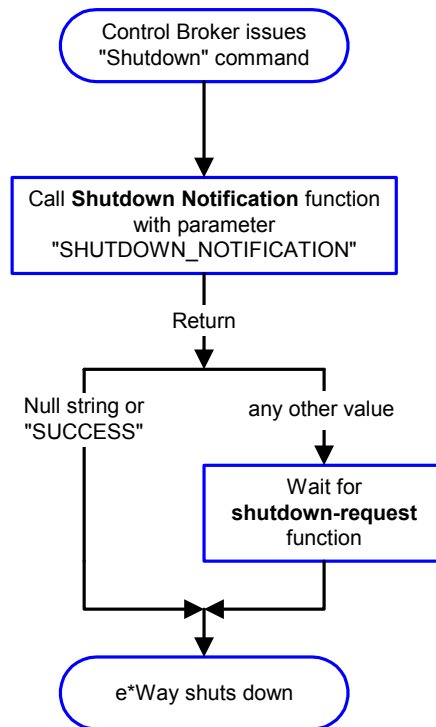
Event-driven Data Exchange Functions

Figure 9 on the next page illustrates event-driven data-exchange using the **Process Outgoing Message Function**.

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

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

Figure 9 Event-driven Data-exchange Functions

```
                        ┌──────────────────────┐
                        │ Collaboration publishes │
                        │   to <EXTERNAL>      │
                        └──────────────────────┘
                                  │
                                  ▼
    ┌──────────────────────┐                         ┌──────────────┐
    │ Call Process Outgoing │◄────────────────────────│  Increment   │
    │   Message function    │                         │   "Resend"   │
    └──────────────────────┘                         │   counter    │
                                                      └──────────────┘
  Null      CONNERR      DATAERR        RESEND                 ▲
  string                                                       │
                                                      ┌──────────────┐
    ┌────────────┐   ┌──────────────┐                 │ Wait for Resend │
    │ Set internal │  │ Increment "Failed │            │ Timeout period │
    │ flag         │  │ Message" counter  │            └──────────────┘
    │ "Connection  │  └──────────────┘                        ▲
    │ Down"        │         │                                │ No
    └────────────┘          ▼                        ◇
                        ◇ Journal              ◇ Maximum
              No        enabled?      No ──── Resends per Message
                                             exceeded?
                            │ Yes                    │ Yes
                            ▼
                    ┌──────────────┐
                    │ Create journal │
                    │     entry      │
                    └──────────────┘
                            │
                            ▼
                    ┌──────────────┐
                    │ Roll back Event │◄──────
                    │ to its publishing │
                    │      IQ         │
                    └──────────────┘
                            │
                            ▼
                        ◯ End
```
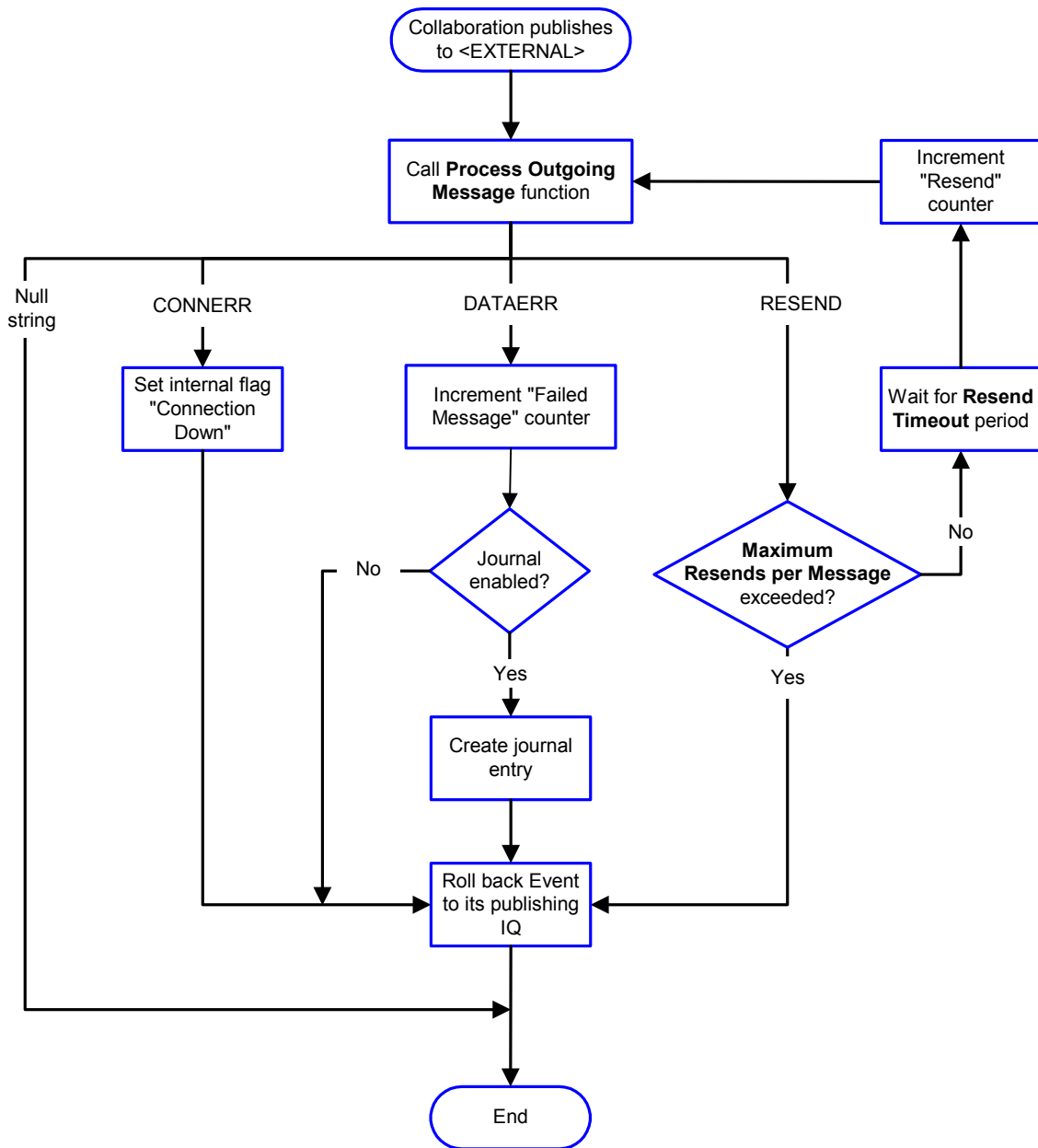
How to Specify Function Names or File Names

Parameters that require the name of a Monk function accepts 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

3.2   COM/DCOM Basics

The Microsoft *Component Object Model* (COM) is a component software architecture that allows applications and systems to be built using separate components. COM is the underlying architecture that forms the foundation for higher-level software services, like those provided by OLE (Object Linking and Embedding). OLE services span various aspects of component software, including compound documents, custom controls, inter-application scripting, data transfer, and other software interactions. By using COM, software objects can be reused for a variety of applications. Because of its binary standard, COM allows any two components to communicate regardless of the language the components are written in.

The Microsoft *Distributed Component Object Model* (DCOM) is an extension of COM, and supports communication among objects on different computers: LANs, WANs, and the Internet. With DCOM, these software objects can be reused over a distributed environment.

Components, or COM objects, are individual modular software routines that can be reused within applications. COM objects are reusable compiled binary objects, as opposed to reusable sections of code. The COM objects create *handles* that provide access to the COM-enabled applications.

One of the standard interfaces exposed by many applications and user-created objects is the IDispatch or "automation" interface. Windows NT and Windows 2000 expose many system services via the automation interface as do several of the Microsoft Office products like Excel. Additionally, the user can also create their own objects using programming languages like C, C++, java, Visual Basic, COBOL, etc. The COM/DCOM e*Way provides access to these objects using the IDispatch interface and semantics. The e*Way can dynamically create an instance of one of these components and then call methods on it.

3.2.1. COM/DCOM e*Way Overview

The COM/DCOM e*Way enables the e*Gate system to exchange data with the server side of COM-enabled applications. The basic architecture of the COM/DCOM e*Way matches the description of a typical e*Way in **Figure 2 on page 11**, with two exceptions:

- ▪ *Operation System Implementation* - the COM e*Way is only available for Windows NT.

- ▪ *No ACK/NAK to External System* - Although the e*Way generates ACKs and NAKs to be queued in e*Gate, the COM/DCOM e*Way does not ACK/NAK the external system.

3.2.2. The Executable and Configuration Files

The executable file for the COM/DCOM e*Way is **stcewgenericmonk.exe**. The default parameter settings for the COM/DCOM e*Way configuration files are contained in the file **stcewmscom.def**, which is stored in the **configs\stcewgenericmonk** directory. This **.def** file should not be modified, since it provides the template for each new

configuration. When choosing a template during the e*Way configuration process, choose **stcewmscom**.

3.2.3. COM/DCOM e*Way Interaction with the External System

The COM e*Way acts like a client to a COM-enabled application or component. While the COM/DCOM e*Way makes a call to the external system, it is directing a request to the specific COM application specified in the e*Way's configuration file by the program identifier (ProgID). Refer to **"ProgID" on page 23**. The external COM application returns a COM *object handle* used to "plug in" the COM application. This handle is used for the remainder of the data exchange session and released upon completion.

Figure 10   Interaction with the External System
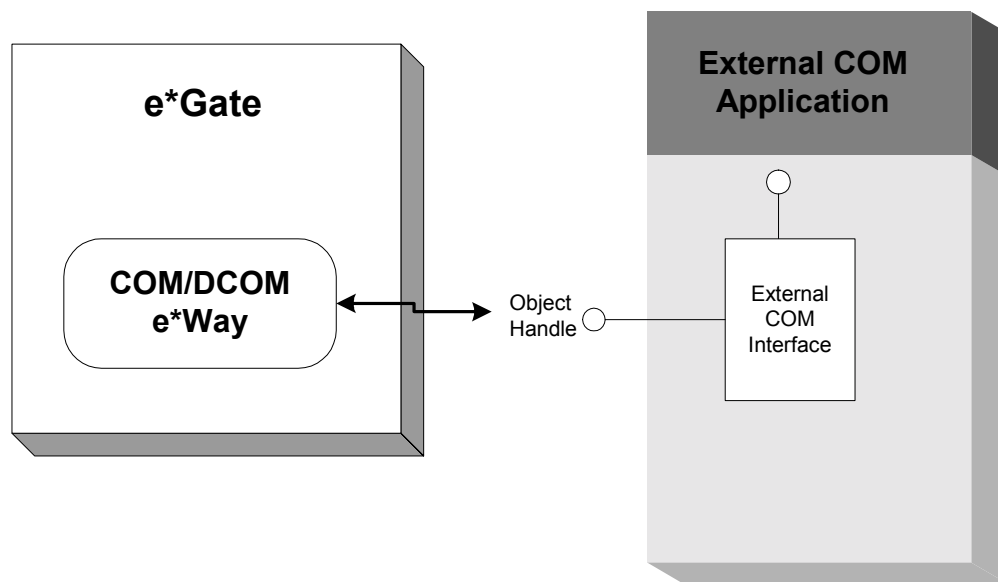
Once the connection to the external COM application has been established, the COM/DCOM e*Way can take advantage of all the functionality that has been made available by the COM application. This makes it possible to access features that are native to the external application, such as worksheet or document functions. This type of access to the external COM application makes it possible to perform virtually any function that you could do within the application itself.

Outbound COM/DCOM e*Way

Figure 11 on the next page and the numbered steps illustrate how an outbound COM/DCOM e*Way interacts with e*Gate and the external COM application.

Figure 11 Outbound COM/DCOM e*Way
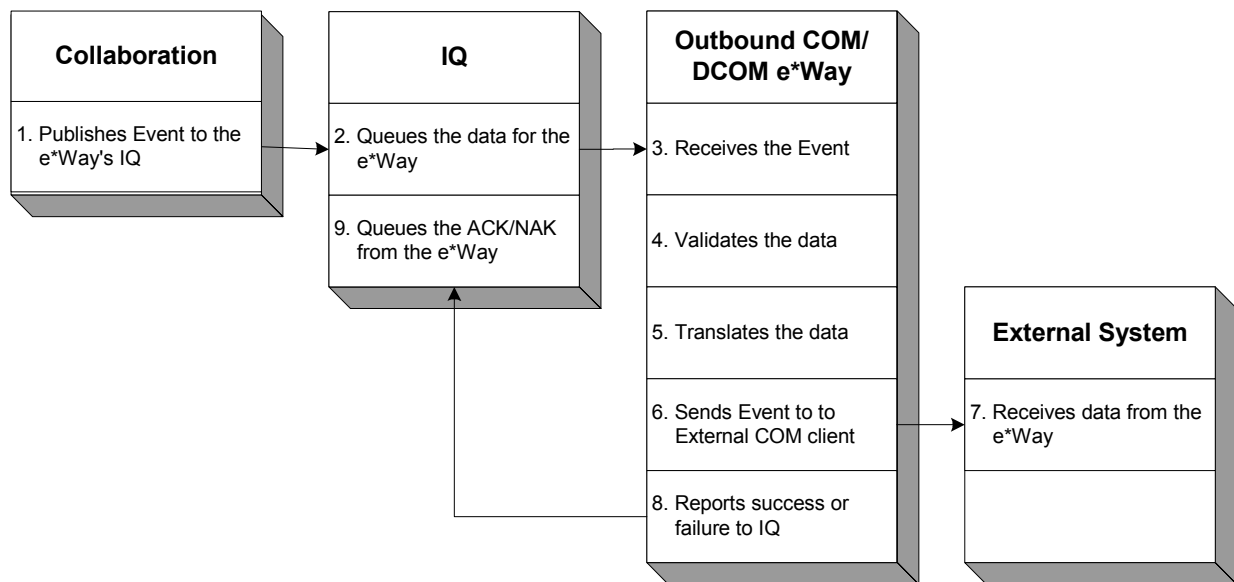
1 The Event is published to the IQ.

2 The IQ stores the data to be used by the other e*Gate components.

3 The outbound COM/DCOM e*Way receives the Event from the IQ.

4 The e*Way ensures that the Event is a valid COM event.

5 The e*Way translates the Event into data that is compatible with the external COM client.

6 The e*Way sends the translated data to the external COM client.

7 The external COM client receives the data from the COM/DCOM e*Way. The communication with the e*Way is synchronous, so errors can be negotiated between the e*Way and the COM client as the data is transferred. This eliminates the need for an ACK/NAK scenario.

8 The e*Way reports the success or failure of the interchange with the COM client to the IQ.

9 The e*Way queues the ACK/NAK reporting the success or failure of the data exchange between the e*Way and the external COM client.

Inbound COM/DCOM e*Way

The following graphic and numbered steps illustrate how an inbound COM/DCOM e*Way interacts with e*Gate and the external COM application.
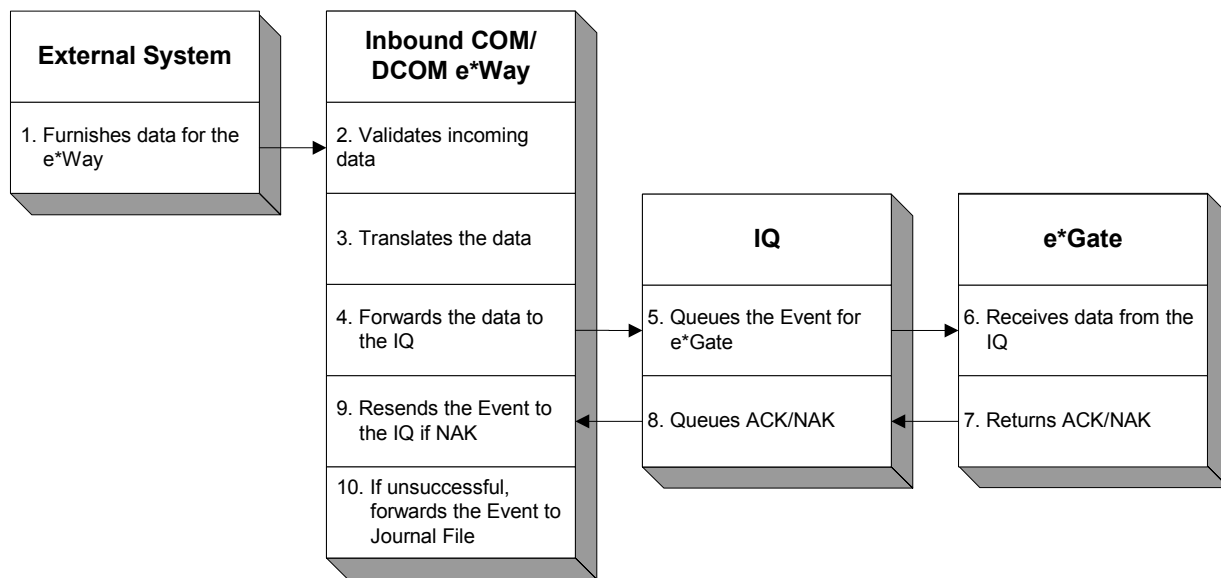
Figure 12 Inbound COM/DCOM e*Way

1   The external system is polled by the inbound COM e*Way at configured intervals. The external system furnishes the data to the inbound COM e*Way.

2   The inbound COM e*Way ensures that the data furnished by the COM client is valid.

3   The inbound COM e*Way then performs any necessary translations to ensure that the data will be useful to the receiving clients.

4   The data is then published to the IQ.

5   The IQ stores the data for receiving e*Ways or Business Object Brokers (BOBs).

6   e*Gate receives the data from the IQ.

7   e*Gate returns either an ACK or a NAK to signal to the inbound COM e*Way that the data was received successfully.

8   The IQ stores the ACKs or NAKs.

9   For a NAK, the inbound COM/DCOM e*Way re-sends the Event to the IQ.

10   If the inbound COM/DCOM e*Way detects excessive NAKs, the Event is written to the Journal File.

3.2.4. COM/DCOM e*Way Functions

Specific Monk functions are available for use in creating and calling COM objects. These functions include:

co-create-instance on page 59

com-invoke on page 60

com-struct-call on page 62

com-startup on page 61

In order to make the initial connection, you must know the name of the server if you are connecting to a DCOM application that does not reside on the local host. Additionally, you must know the program identifier (ProgID) for the application.

ProgID

The ProgID is a Windows registry entry that uniquely identifies a program or a COM object. Unlike a Globally Unique Identifier (GUID), the ProgID is a humanly-readable alphanumeric string. The ProgID consists of three parts in this format:

```
vendor.component.version
```

The *vendor* parameter is the control's library, the *component*, it's class, and the *version* number, which is optional. If no version is given, the latest version of the application object is assumed. Each parameter is separated by periods and no spaces; for example the Visual Basic Command Button object's ProgID is:

```
VB.CommandButton
```

The ProgID for most objects can be determined by checking, for example, the Visual Basic Object Browser. The Windows registry entry reads as follows:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\ProgID =
```

The following tables lists standardized ProgIDs for ActiveX Controls and Microsoft applications:

Table 2   ActiveX Standard ProgIDs

Control	Identifier
CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

Table 3   Microsoft Access Standard ProgIDs

Object	Identifier
Application	Access.Application, Access.Application.9

Table 3 Microsoft Access Standard ProgIDs

Object	Identifier
CurrentData	Access.CodeData, Access.CurrentData
CurrentProject	Access.CodeProject, Access.CurrentProject
DefaultWebOptions	Access.DefaultWebOptions

Table 4 Microsoft Excel Standard ProgIDs

Object	Identifier	Comments
Application	Excel.Application, Excel.Application.9	
Workbook	Excel.AddIn	
Workbook	Excel.Chart, Excel.Chart.8	Returns a workbook containing two worksheets; one for the chart and one for its data. The chart worksheet is the active worksheet.
Workbook	Excel.Sheet, Excel.Sheet.8	Returns a workbook with one worksheet.

Table 5 Microsoft Graph Standard ProgIDs

Object	Identifier
Application	MSGraph.Application, MSGraph.Application.8
Chart	MSGraph.Chart, MSGraph.Chart.8

Table 6 Microsoft Office Web Components Standard ProgIDs

Object	Identifier
ChartSpace	OWC.Chart, OWC.Chart.9
DataSourceControl	OWC.DataSourceControl, OWC.DataSourceControl.9
ExpandControl	OWC.ExpandControl, OWC.ExpandControl.9
PivotTable	OWC.PivotTable, OWC.PivotTable.9
RecordNavigationControl	OWC.RecordNavigationControl, OWC.RecordNavigationControl.9
Spreadsheet	OWC.Spreadsheet, OWC.Spreadsheet.9

Table 7 Microsoft Outlook Standard ProgIDs

Object	Identifier
Application	Outlook.Application, Outlook.Application.9

Table 8 Microsoft PowerPoint Standard ProgIDs

Object	Identifier
Application	PowerPoint.Application, PowerPoint.Application.9

Table 9 Microsoft Word Standard ProgIDs

Object	Identifier
Application	Word.Application, Word.Application.9
Document	Word.Document, Word.Document.9, Word.Template.8
Global	Word.Global

Invoking Methods from a COM Object Interface using Monk

Once a COM object or program is called, the object interface methods associated with that object are available for use. Refer to the documentation for application you are connecting to for a list of exposed methods.

The following pseudo code example shows how to connect to an object, obtain object handles, and invoke methods using Monk. See **"COM/DCOM Functions" on page 58** for a complete description of the Monk functions used. **Figure 13 on page 26** illustrates the process.

1. Make Connection to the Application Object

```
(define Appl_HDL (co-create-instance progid))
```

The result is a handle to the application object. All application methods are now available for use.

2. Invoke the Business Object Handle

```
(define BO_HDL (com-invoke Appl_HDL "GetBusObject" "FUNC" '#()))
```

The result is a handle to the business object. Methods on the object represented by progid are now available.
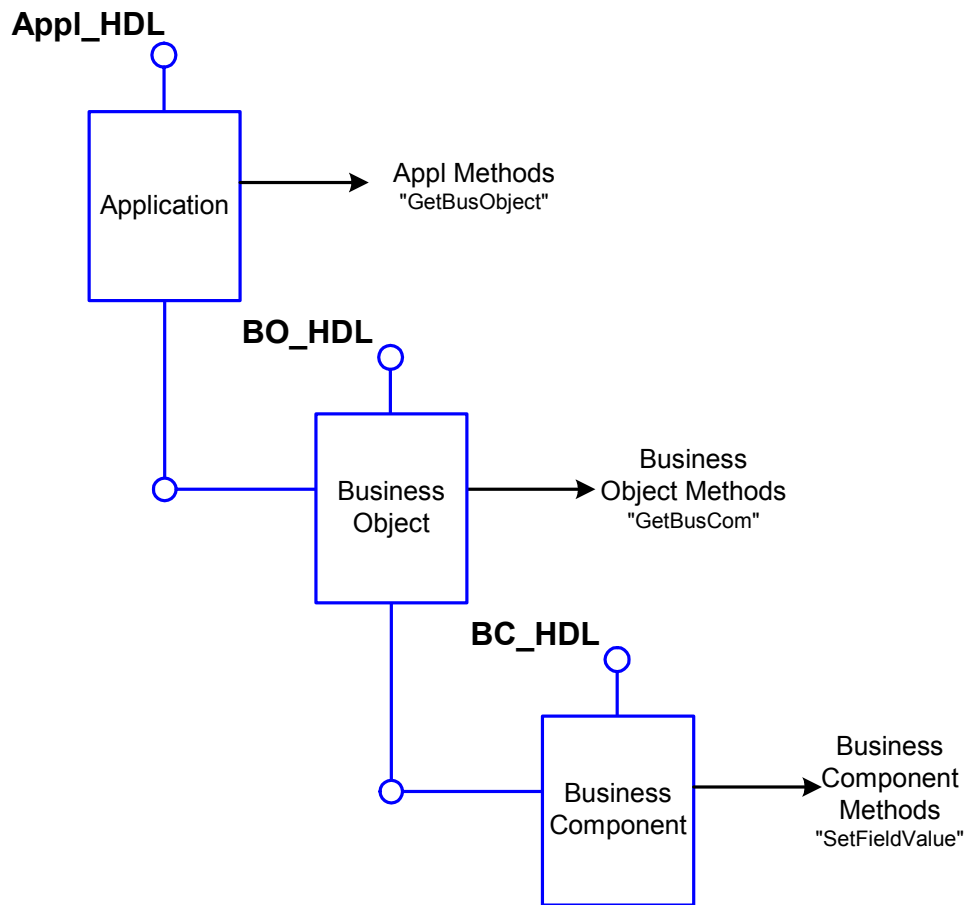
3. Invoke the Business Component Object

```
(define BC_HDL (com-invoke BO_HDL "GetBusCom" "FUNC" '#()))
```

The result is a handle to the business component object. All business component object methods are now available for use.

4. Invoke the Business Component Object Method

```
(com-invoke BC_HDL "setFieldValue" "FUNC" Param))
```

Figure 13   COM Object Methods and Handles

Configuration

This chapter describes how to configure the COM/DCOM e*Way by setting the configuration parameters using the e*Way Editor.

This Chapter Explains:

4.1 e*Way Configuration Parameters

COM/DCOM e*Way configuration parameters are set using the e*Way Editor.

To change e*Way configuration parameters:

1 In the Enterprise Manager's Component editor, select the e*Way you want to configure and display its properties.

2 Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.

*Note: When creating a new e*Way, you must also select the **stcewmscom** template file from the **e*Way Template Selection** list.*

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* chapter in the *e*Gate Integrator User's Guide*.

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

- General Settings
- Communication Setup
- Monk Configuration

- COM Settings

4.1.1. General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is 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 is 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 29** for more information.)

Max Resends Per Message

Description

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

Required Values

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

Max Failed Messages

Description

Specifies the maximum number of failed messages (Events) that the e*Way allows. When the specified number of failed messages is reached, the e*Way shut downs and exits.

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 is queued to the e*Way's configured queue. If this parameter is set to **No**, then error messages will be ignored. See **"Exchange Data with External Function" on page 34** for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages is not forwarded. See **Figure 7 on page 16** for more information about how the e*Way uses this function.

4.1.2. 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 (**Start Up** tab) in the Enterprise Manager controls when the e*Way executable runs. The schedule you set within the parameters discussed in this section (using the e*Way Editor) determines when data is exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

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

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

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

If you do not do so, the e*Way terminates 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 Function** and **Negative Acknowledgement Function**) 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** is 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 34**, **"Exchange Data Interval" on page 30**, and **"Stop Exchange Data Schedule" on page 30** 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 is ignored and the e*Way invokes the **Exchange Data with External Function** immediately.

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

See **"Down Timeout" on page 30** and **"Stop Exchange Data Schedule" on page 30** for more information about the data-exchange schedule.

Down Timeout

Description

Specifies the number of seconds that the e*Way waits between calls to the **External Connection Establishment Function**. See **"External Connection Establishment Function" on page 35** 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 waits between calls to the **External Connection Verification Function**. See **"External Connection Verification Function" on page 35** 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 waits between attempts to resend a message (Event) to the external system, after receiving an error message from the external system.

Required Values

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

Zero Wait Between Successful Exchanges

Description

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

Required Values

Yes or **No**. The default is **No**.

Additional Information

If this parameter is set to **Yes** and the previous exchange function returned data, then the e*Way immediately invokes the **Exchange Data With External Function**. If this parameter is set to **No**, the e*Way always waits the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External Function**.

See **"Exchange Data with External Function" on page 34** for more information.

A series of figures in **Chapter 3** illustrate the interaction and operation of these functions.

4.1.3. 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.

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** is searched after the default load path.

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 are automatically 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.

This parameter is optional and may be left blank.

Monk Environment Initialization File

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

Required Values

A filename within the "load path", or filename plus path information (relative or absolute). If path information is specified, that path is appended to the "load path." See **"Additional Path" on page 32** 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 loads this file and tries to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way would attempt to execute the function **my-init**).

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

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 4 on page 13**).

Startup Function

Description

Specifies a Monk function that the e*Way loads and invokes 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 is called after the e*Way loads the specified **Monk Environment Initialization File** and any files within the specified **Auxiliary Library Directories**.

The e*Way loads this file and tries to invoke a function of the same base name as the file name (see **Figure 4 on page 13**). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

Process Outgoing Message Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. *You may not leave this field blank.*

Additional Information

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

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Enterprise Manager). The function returns one of the following (see **Figure 7 on page 16** 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.

Note: *If you wish to use **event-send-to-egate** to enqueue failed Events in a separate IQ, the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See **"event-send-to-egate" on page 52** for more information.*

Exchange Data with External Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is optional and may be left blank. However, this parameter is required if a schedule was set using the **Start Exchange Data Schedule** parameter. If so, you must also define the following:

- **Positive Acknowledgement Function**
- **Negative Acknowledgement Function**

Additional Information

The function accepts no input and must return a string (see **Figure 9 on page 18** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information is 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.

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

This function is initially triggered by the **Start Exchange Data Schedule** or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been ACKed or NAKed (by the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter. If this parameter is set to **Yes**, the e*Way immediately calls the **Exchange Data with External Function** again; otherwise, the e*Way does not call the function until the next scheduled "start exchange" time or the schedule is manually invoked using the Monk function **start-schedule** (see **start-schedule** on page 57 for more information).

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way calls when it has determined that the connection to the external system is down (or is unknown).

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 **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 calls 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 executes 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 **External Connection Establishment Function** is called when the e*Way has determined that its connection to the external system is down or is unknown.

External Connection Shutdown Function

Description

Specifies a Monk function that the e*Way calls 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 is only invoked when the e*Way receives a "suspend" command from a Control Broker. When the "suspend" command is received, the e*Way invokes this function, and passes the string "SUSPEND_NOTIFICATION" as an argument.

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

Positive Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls 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** is called again, with the same input data.

- Null string: The function completed execution successfully.

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

Negative Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls 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 is called again.

- Null string: The function completed execution successfully.

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

Shutdown Command Notification Function

Description

Specifies a Monk function that is 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 calls 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 does not proceed until the Monk function **shutdown-request** is executed (see **shutdown-request** on page 56).

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.*

4.1.4. COM Settings

The parameters in this section help you to set up the required information for the COM/DCOM e*Way.

ProgID

Description

Specifies the *Program ID* of the COM-enabled application to connect to. Refer to **"ProgID" on page 23** for more information on application-specific identifiers.

Required Values

A program ID.

Server

Description

Specifies the server name (hostname) where the COM object resides. Only required if the COM object does not reside on the local host. This is only applicable to EXEs.

Required Values

A server name.

Additional Information

Either an alias or IP addresses.

Implementation

This chapter provides information on how to implement the COM/DCOM e*Way using the ETD Editor. Also included is a sample implementation represented by an output file.

This Chapter Explains:

- **"Implementation Notes" on page 39**
- **"Using the ETD Editor's Build Tool" on page 41**
- **"The Sample Implementation" on page 43**

5.1 Implementation Notes

The COM/DCOM e*Way supports the following data types:

Table 10 Supported Data Types

OLE Data Types	Description	Return Value
VT_I2	2-byte signed int	Yes
VT_I4	4-byte signed int	Yes
VT_R4	4-byte real	Yes
VT_R8	8-byte real	Yes
VT_BSTR	Binary string	Yes
VT_DISPATCH	IDispatch	Yes
VT_ERROR	4-byte error code	Yes
VT_BOOL	Boolean	Yes
VT_I1	1-byte	Yes
VT_UI1	Unsigned char	Yes
VT_UI2	Unsigned short	Yes
VT_UI4	Unsigned int	Yes
VT_INT	Int	Yes
VT_UINT	Unsigned int	Yes

Table 10   Supported Data Types (Continued)

OLE Data Types	Description	Return Value
VT_SAFEARRAY ▪ VT_I2 ▪ VT_I4 ▪ VT_R4 ▪ VT_BSTR ▪ VT_DISPATCH ▪ VT_ERROR ▪ VT_BOOL ▪ VT_UI1	See **Note** below.	Yes

All OLE data types are supported by "in" behavior and all OLE data types except for VT_SAFEARRAY are supported by "out" behavior.

Note: ***The code will process one or two dimensional SAFEARRAYSs only! The number of elements in all rows is expected to be equal!*** VT_SAFEARRAY has specific, limited functionality as used in the COM/DCOM e*Way--see the following subsection for appropriate SAFEARRAY use.

5.1.1. Appropriate SAFEARRAY Use

In COM, the SAFEARRAY type can be multi-dimensional (e.g.: int array[5][10][15], an array of arrays of arrays) and, further, the different arrays may have differing lower bounds indices. The **COM/DCOM e*Way**, however, must follow these conventions:

▪ The e*Way can only support a *maximum* of **two dimensions** (e.g.: int array[2][5]).

▪ Each array (if two dimensions) must have the same number of subarrays.

▪ The index numbering must be identical.

Using zero-based indices, the SAFEARRAY type can be graphically viewed as:

```
array[0]
    subarray[0]
    subarray[1]
    subarray[2]
    subarray[3]
    subarray[4]
array[1]
    subarray[0]
    subarray[1]
    subarray[2]
    subarray[3]
    subarray[4]
```

The above illustrates the requirement that arrays be limited to a maximum of two dimensions, that the number of subarrays be equal, and that the index numbering for both arrays be identical.

5.2 Using the ETD Editor's Build Tool

In COM, the event type corresponds to the object being used; it's properties and methods. These properties and methods are exposed in the collaboration so you can get/set the property or call one or more methods.

The Event Type Definition Editor's Build tool automatically creates an Event Type Definition file based upon sample data. Use this procedure to create an Event Type Definition based upon the data your installation requires.

To create an Event Type Definition using the Build tool:

1 Launch the Event Type Definition Editor.

2 On the ETD Editor's Toolbar, click **Build**.

The **Build an Event Type Definition** dialog box appears.

Figure 14 Build an Event Type
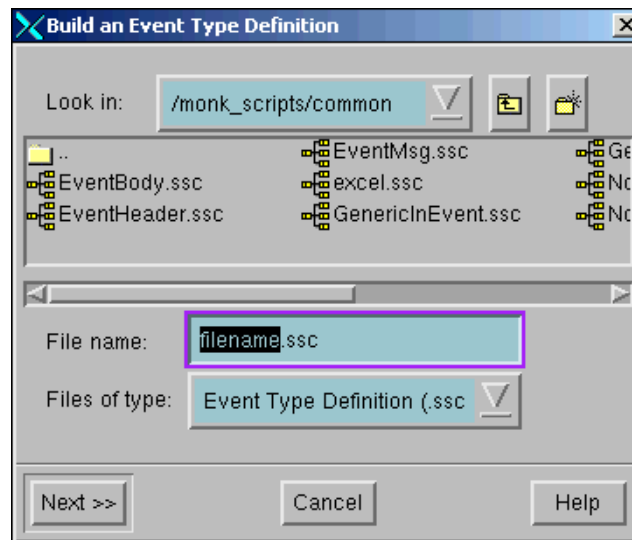
3 In the **File Name** box, type the name of the ETD file to build. *Do not specify any file extension*—the Editor supplies an "ssc" extension for you.

4 Click **Next**.

Figure 15 Build an Event Type (continued)
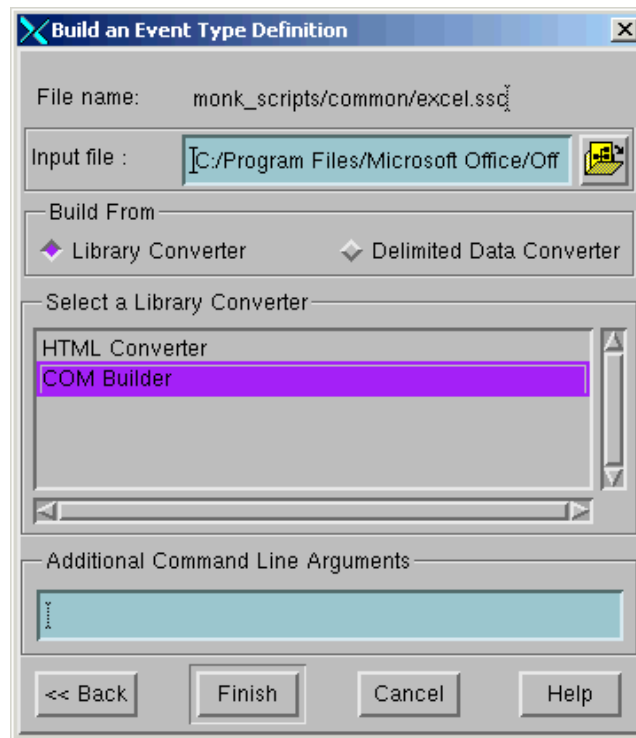
5   In the **Input file** box, type the name of the input file, using an **.olb** or **.tlb** extension.

Note: *Note: COM type library files describe the methods and properties exposed from an automation compatible component. Com type library typically have the file extension .tlb or .olb. However, most components embed the type library file in the DLL, OCX or EXE that houses the component.*

6   Under **Build From**, select **Library Converter**.

7   Under **Select a Library Converter**, select **COM Builder**.

8   In the **Additional Command Line Arguments** box, type any additional arguments.

9   Click **Finish.**

To create an Event Type Definition from the command line:

Create an Event Type Definition file using **stccombuilder.exe** by typing the following at the command prompt:

```
stccombuilder -eg -rh registry -rp port -un username -up password
-rs schema -tf temp_file -i input_file output_file
```

where

registry is the name of the Registry Host

port is a valid port number (such as 4000)

username and *password* are a valid e*Gate username/password combination

schema is the name of the schema you wish to register the ETD file to

temp_file is the name of a temporary file created and deleted by **stccombuilder.exe**

input_file is the name of the **.olb** or **.tlb** input library file, and

output_file is the name of the **.ssc** file where the Event Type Definition is written.

5.3   The Sample Implementation

A sample of the COM e*Way is included on the Installation CD-ROM. The sample demonstrates e*Gate's interaction with a COM enabled application—Microsoft Excel.

5.3.1. Sample Schema Overview

The sample provided on the Installation CD is called "ComSample.zip". This sample schema opens a Microsoft Excel worksheet and populates the first three rows with text strings. This schema requires Excel to be present on the local machine.

The sample includes an inbound file e*Way, an IQ, an outbound COM e*Way, and all of the components required to run the scenario. The input file e*Way reads an input file as a way of generating the initial Event; the content of the input file is irrelevant to the schema. The outbound COM e*Way writes the output to a worksheet: **c:\temp\test.xls** using the logic written into **excel.tsc**. (See **Figure 16 on page 43**).

Figure 16   Sample Schema
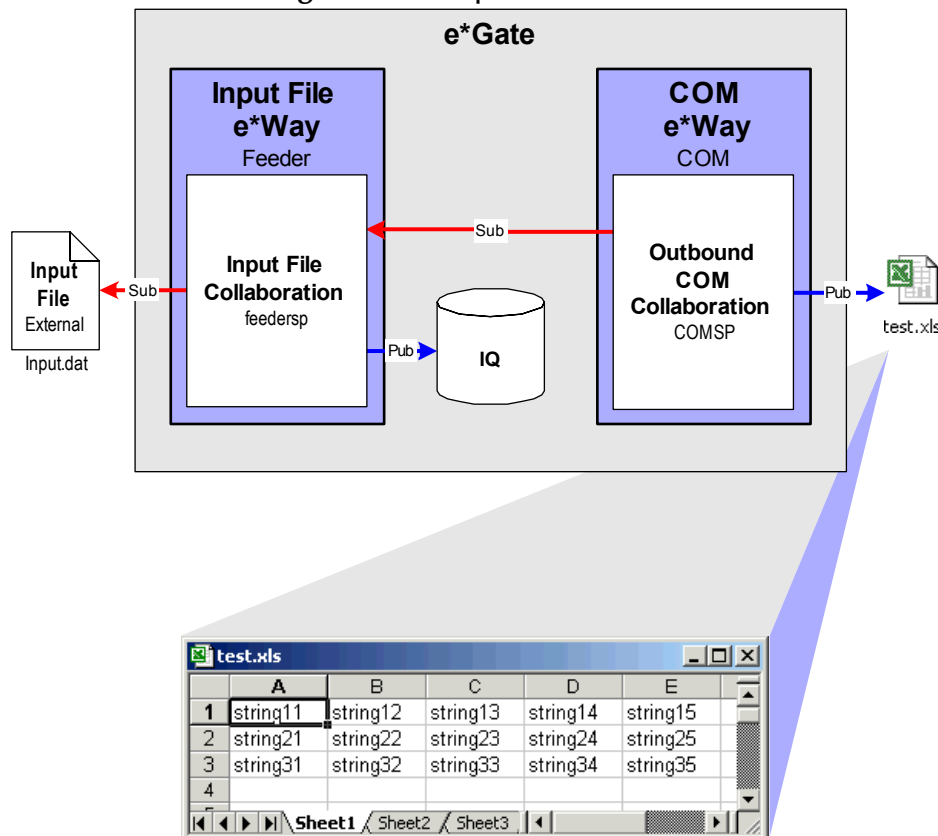
5.3.2. # Installing the Sample Schema

The ComSample.zip sample files are located on the Installation CD-ROM in the **\samples\ewmscom** directory.

Importing the Sample Schema

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 COM/DCOM e*Way, do the following:

1 Start the e*Gate Enterprise Manager GUI.

2 When the Enterprise Manager 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 ComSample, or any appropriate name as desired.

5 Select **Create from export**, locate the **ComSample.zip** on the CD and click **Open**. The e*Gate Enterprise Manager opens to the sample schema.

Complete the Configuration of the Sample Schema

The ComSample schema is nearly ready to use as it is presently configured. The only change to be made is to update the settings of the **Feeder** e*Way.

To change the configuration of the **Feeder** e*Way:

1 From the e*Gate Enterprise Manager, navigate to the **Feeder** e*Way.

2 Click ⬜ to display the e*Way's properties.

3 Click **Edit** to open the e*Way Editor.

4 Select **Poller (inbound) settings** from the list of parameter sections.

5 Select **InputFileMask** from the list of parameters.

6 Enter ***.dat** and click ⬜ to add the item.

7 From the **File** menu, choose **Promote to Run Time...**

8 Click **OK** to promote the file and **OK** again to close the editor.

9 Click **OK** to close the e*Way Properties.

Prepare the Input and Output Files

The sample schema requires two files in order to run: **c:\temp\InputFile.dat** (the input file) and **c:\temp\test.xls** (the output file).

To create the input file

1 Open any text editor, such as Windows Notepad.

2 Create a new file called **InputFile.dat**

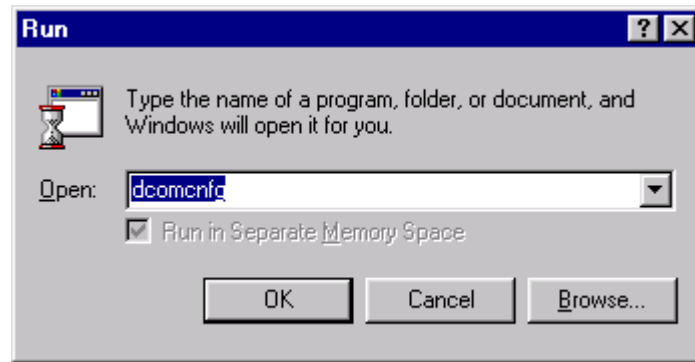   **3** Save this file to the **c:\temp** directory and close the text editor.

To create the output file

    1 Open Microsoft Excel.

    2 Create a new worksheet called **test.xls**.

    3 Save this file to the **c:\temp** directory and close Excel.
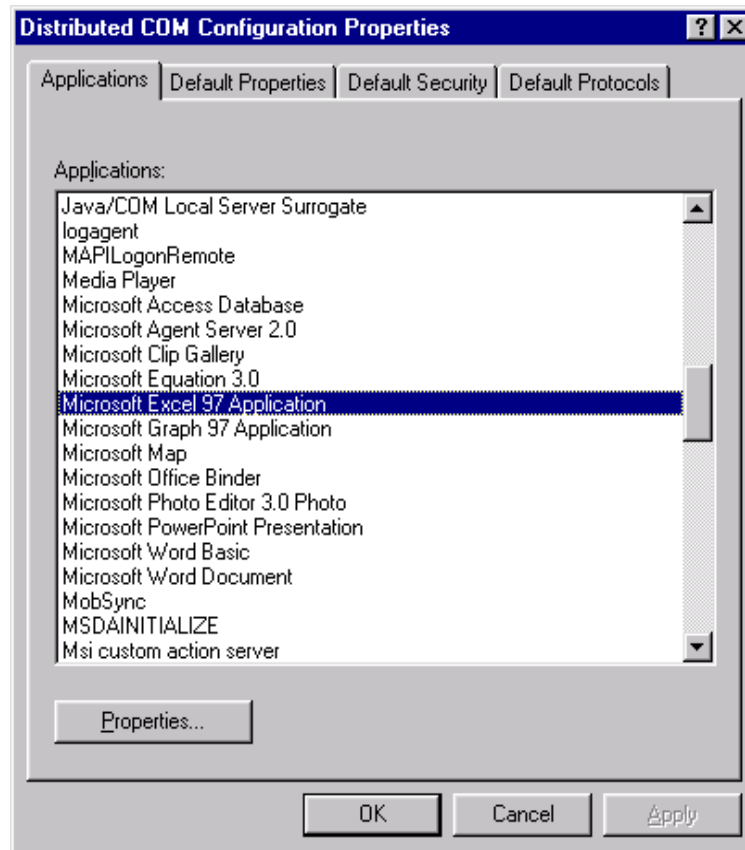
5.3.3. Configuring the DCOM Server

1 To configure the DCOM Server, in the **Run** dialog type in the command **dcomcnfg** and click **OK**.

Figure 17 Run Dialog

The Distributed COM Configuration Properties dialog lists the registered applications.

Figure 18 Distributed COM Configuration Properties

2 Select the **Microsoft Excel** application.

3   Click **Properties**.

4   Select the **Security** tab.

Figure 19   Application Properties - Security Tab
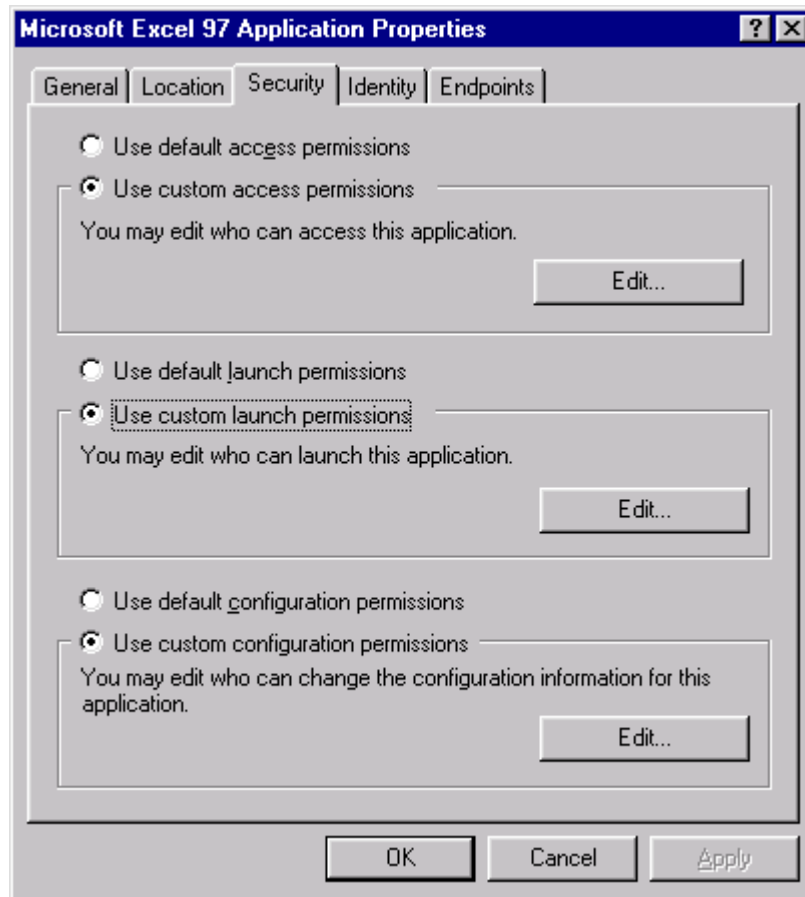
5   Select the **Use custom access permission** option button and click **Edit**.

Figure 20 Add Users and Groups - Access
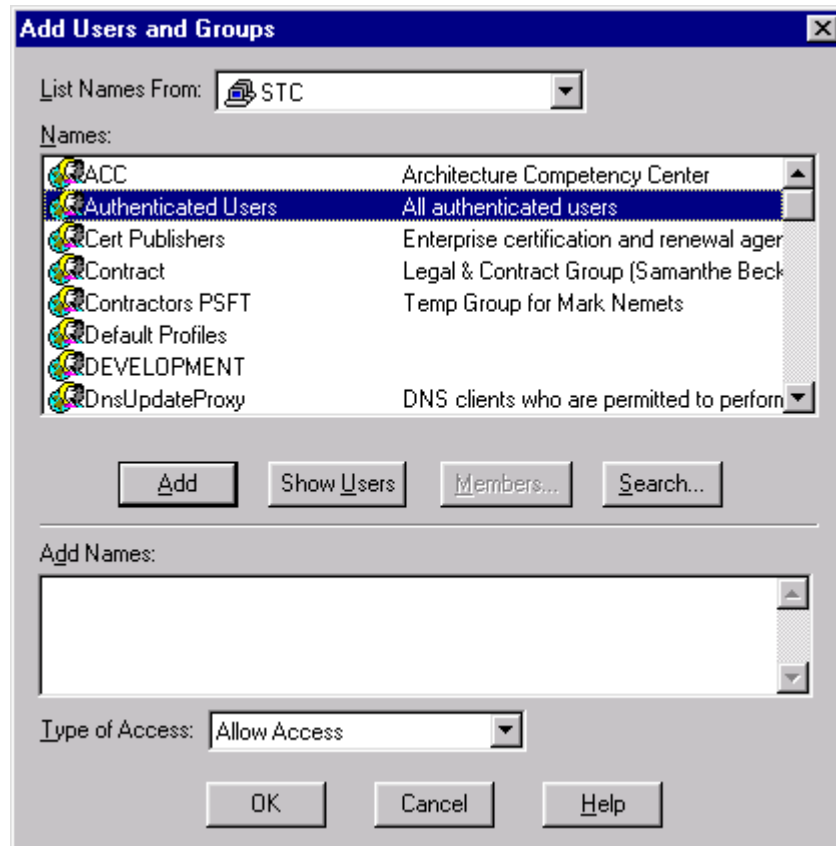
6   Select the appropriate users to whom you want to grant access; for example, **Authenticated Users**.

7   Click **Add**.

8   For Type of Access, select **Allow Access**.

9   Click **OK**.

10 In the Application Properties window (see Figure 19), select the **Use custom launch permission** option button and click **Edit**.

11 Select the appropriate users to whom you want to grant launch permission; for example, **Authenticated Users**.

12 Click **Add**.

13 For Type of Access, select **Allow Launch**.

14 Click **OK**.

15 In the Application Properties window (see Figure 19), select the **Identity** tab.

Figure 21 Application Properties - Identity Tab
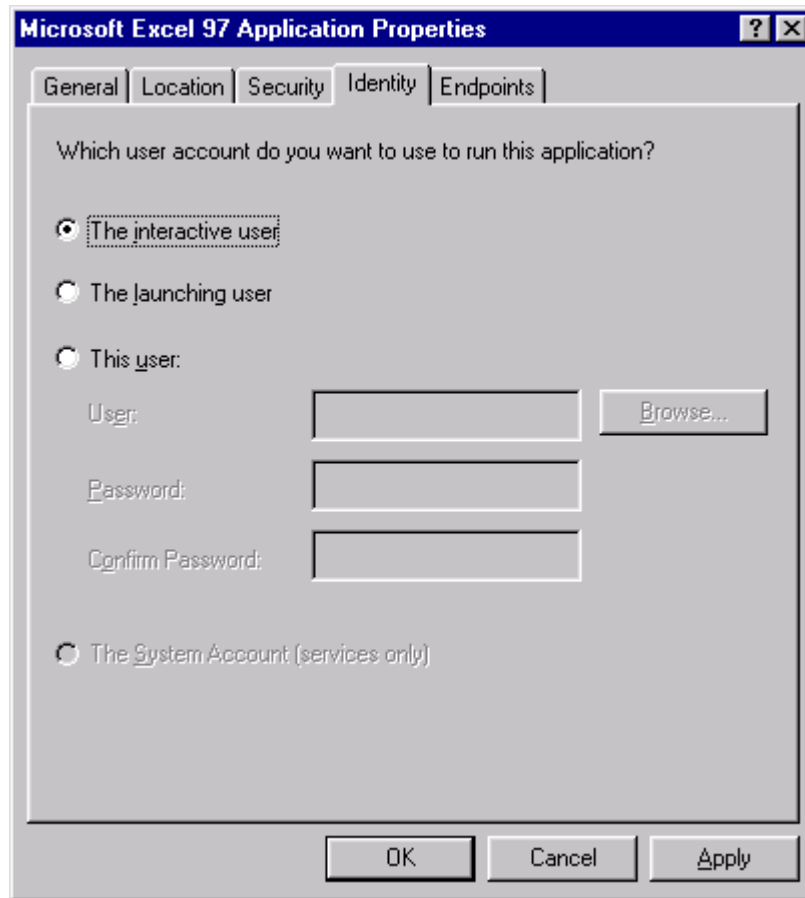
16 Click **The interactive user** option button, then **OK**.

The DCOM server is now configured for the sample schema.

5.3.4 **Importing the Sample Schema**

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 COM/DCOM e*Way, do the following:

1 Start the e*Gate Enterprise Manager GUI.

2 When the Enterprise Manager 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 **COMtest**, or any appropriate
name as desired.

5 Select **Create from export**, locate the **ComSample.zip** on the CD and click **Open**.
The e*Gate Enterprise Manager opens to your new schema.

5.3.5. **Running the Sample Schema**

To Run the ComSample Schema:

Type the following at the command prompt:

```
stccb -ln logical_name -rh registry -rs ComSample -un username
-up password
```

where

logical_name is the name of the Control Broker,

registry is the name of the Registry Host, and

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

After a few moments, the input file will be loaded and queued as an e*Gate Event, the COM e*Way will be executed, and a COM session with **test.xls** will be initiated. A few moments later, **test.xls** will be displayed and first five columns of the first five rows will be populated with text.

COM/DCOM e*Way Functions

This chapter describes the functions used by the COM/DCOM e*Way. These functions can only be used by the functions defined within the e*Way's configuration file. None of the functions are available to Collaboration Rules scripts executed by the e*Way.

The COM/DCOM e*Way's functions fall into the following categories:

- **Basic Functions** on page 51
- **COM/DCOM Functions** on page 58

6.1 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 52

get-logical-name on page 53

send-external-down on page 54

send-external-up on page 55

shutdown-request on page 56

start-schedule on page 57

stop-schedule on page 58

event-send-to-egate

Syntax

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

Description

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

Parameters

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

Return Values

Boolean
Returns **#t** if the data is sent successfully; otherwise, returns **#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 Enterprise Manager).

Throws

None.

send-external-down

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown request requests the e*Way to perform the shutdown procedure when there is no outstanding incoming/outgoing event. When the e*Way is ready to act on the shutdown request, in invokes the **Shutdown Command Notification Function** (see **"Shutdown Command Notification Function" on page 37**). Once this function is called, the shutdown proceeds immediately.

Parameters

None.

Return Values

None.

Throws

None.

start-schedule

Syntax

```
(start-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

6.2 COM/DCOM Functions

This section provides a summary of the COM/DCOM e*Way functions which are:

co-create-instance

Syntax

```
(co-create-instance progid server)
or
(co-create-instance progid)
```

Description

co-create-instance creates an instance of a COM object as specified in the Registry.

Parameters

Name	Type	Description
progid	string	Programmatic identifier for the COM object or application. See **"ProgID" on page 23** for more information.
server	string	Name of the server where the COM object resides. Only required if the COM object is an EXE and does not reside on the local host.

Return Value

handle

Returns a handle to an object upon success.

Boolean

Returns **#f** and displays the error message if the creation fails.

Throws

None.

Location

None.

Example

```
(define com-appl-handle (co-create-instance "obj.obj"))
```

com-invoke

Syntax

```
(com-invoke obj-handle function-name function-type
vector-of-parameters)
```

Description

com-invoke calls the method of the object with the name of *function-name* and parses the *vector of parameters* as input.

Parameters

Name	Type	Description
obj handle	handle	Handle returned by the **co-create-instance** function or from another com-invoke function.
function name	string	Name of the object.
function type	string	Types include: FUNC - function. PROPERTYGET - to obtain properties. PROPERTYPUT - to send properties directly. PROPERTYPUTREF - to send properties by reference optional - as needed by Word, Excel, etc.
vector of parameters	vector	Function parameters presented as vectors. Outside parenthesis represent outer vectors. Inside parenthesis represent inner vectors. Each inner vector represents one parameter or block. Each outer vector represents a set of parameters.

Return Value

Boolean

Returns **#f** if the function fails; otherwise, returns **#t** if it succeeds. The output parameters to the function call is set in the *vector-of-parameters.*

Throws

None.

Location

None.

Example

```
(com-invoke com-appl-handle "FUNCTION1" "FUNC" (VECTOR (VECTOR "in"
"VT_BSTR" "Text" "Testing String")))
```

com-startup

Syntax

```
(com-startup "")
```

Description

com-startup calls the following if the global Monk variable **com-appl-server** is set by the user:

```
(co-create-instance com-appl-progid com-appl-server)
```

Otherwise, it calls:

```
(co-create-instance com-appl-progid)
```

Parameters

None (empty string).

Return Value

None.

Throws

None.

Location

None.

Example

```
(com-startup "")
```

com-struct-call

Syntax

```
(com-struct-call obj-handle method-path)
```

Description

com-struct-call calls the **com-invoke** function.

Parameters

Name	Type	Description
obj-handle	Customized Monk object	Object handle.
method-path	Message path	Interface method node.

Return Value

Monk object
If successful, returns Monk objects representing supported OLE data types.

Boolean
Returns **#f** if failure.

Throws

None.

Location

None.

Example

```
(com-struct-call com-appl-handle ~output%excel._application.Visible-
PUT)
```

where *com-appl-handle* is the object handle to an Excel application, and
~output%excel._application.Visible-PUT is the node path for the COM interface method
on the Monk Structure.

Note: *There are three primitives that are in the .dll file that are not normally used: com-appl-handle, com-appl-progid, and com-appl-server.*

Index