

SeeBeyond ICAN Suite

e*Way Intelligent Adapter for PeopleSoft Batch User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)



The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20050406025037.

Contents

Preface	7
Intended Reader	7
Organization	7
Nomenclature	8
Online Use	8
Writing Conventions	8
Additional Documentation	9
<hr/>	
Chapter 1	
Introduction	10
Overview	10
Batch e*Way	10
ODBC e*Way	10
e*Gate to PeopleSoft	11
PeopleSoft to e*Gate	12
Components	13
Supported Operating Systems	13
<hr/>	
Chapter 2	
Installation	14
System Requirements	14
External System Requirements	15
SOCKS Support	15
OS/390 Support	15
Installing the e*Way	16
Windows Systems	16
Installation Procedure	16
Subdirectories and Files	18
Environment Configuration	20
UNIX Systems	21

Installation Procedure	21
Subdirectories and Files	21
Environment Configuration	24
Optional Example Files	25
Installation Procedure	25
Subdirectories and Files	26

Chapter 3

System Implementation	27
Overview	27
Implementation Sequence	28
Viewing e*Gate Components	28
Creating a Schema	29
Creating Event Types	30
Generating Event Type Definitions	31
Using the ETD Editor's Build Tool	31
Assigning ETDs to Event Types	33
Defining Collaborations	34
Creating Intelligent Queues	35
Sample Schema	35
Subscribing to an External System	35
Publishing to an External System	36

Chapter 4

Setup Procedures	38
Overview	38
Setting Up the e*Way	39
Creating the e*Way	39
Modifying e*Way Properties	40
Configuring the e*Way	41
Using the e*Way Editor	42
Changing the User Name	45
Setting Startup Options or Schedules	45
Activating or Modifying Logging Options	47
Activating or Modifying Monitoring Thresholds	48
Troubleshooting the e*Way	49
Configuration Problems	49
System-related Problems	50

Chapter 5

Operational Overview	51
e*Gate to PeopleSoft	51
Outbound Process Flow	52
PeopleSoft to e*Gate	54
Inbound Process Flow	54
Batch Operation	56
Message-Based Operation	56
General Information	56
Receiving Data with a <i>Receive Order</i>	57
Error Reporting	59
ODBC Operation	60
Using SQL Functions	60
Static vs. Dynamic SQL Functions	60
Benefits of Stored Procedures	60
ODBC SQL Type Support	62
e*Way Architecture	63
Event Type Definitions and Collaborations	64
Basic e*Way Processes	65
Initialization Process	66
Connect to External Process	67
Data Exchange Process	68
Disconnect from External Process	71
Shutdown Process	71

Chapter 6

Configuration Parameters	72
Overview	72
General Settings	73
Communication Setup	75
Monk Configuration	79
Specifying Function or File Names	79
Specifying Multiple Directories	79
Load Path	79
Dynamic Configuration	88

Chapter 7

API Functions	93
Overview	93

Contents

Batch e*Way Standard Functions	94
ODBC e*Way Standard Functions	101
Generic e*Way Functions	113

Appendix A

Document Type Definitions	120
Receive XML Messages	120
Error Messages	121
Data Message	122

Appendix B

ODBC Monk Scripts	124
Sample Monk Scripts	124
Initializing Monk Extensions	125
Calling Stored Procedures	126
Inserting Records with Dynamic SQL Statements	128
Updating Records with Dynamic SQL Statements	130
Selecting Records with Dynamic SQL Statements	132
Deleting Records with Dynamic SQL Statements	134
Inserting a Binary Image to a Database	135
Retrieving an Image from a Database	138
Common Supporting Routines	140

Index	143
--------------	------------

Preface

This Preface contains information regarding the User's Guide itself.

P.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the SeeBeyond™ e*Gate™ Integrator system, and have a working knowledge of:

- Windows 2000 and/or UNIX operations and administration
- Windows-style GUI operations
- FTP and ODBC concepts and operations
- Integrating PeopleSoft Batch applications with external systems

P.2 Organization

This User's Guide is organized roughly into two parts. The first part, consisting of Chapters 1-4, introduces the e*Way and describes the procedures for installing and setting up the e*Way, and implementing a working system incorporating the e*Way. Chapter 3 also contains descriptions of the sample schemas provided with the product. These can be used to test your system following installation and, if appropriate, as templates you can modify to produce your own custom schemas. This part should be of particular interest to a System Administrator or other user charged with the task of getting the system up and running.

The second part, consisting of Chapters 5-8, describes the details of e*Way operation and configuration, including descriptions of the API functions. This part should be of particular interest to a Developer involved in customizing the e*Way for a specific purpose. Information contained in this part that is necessary for the initial setup of the e*Way is cross-referenced in the first part of the guide, at the appropriate points in the procedures.

P.3 Nomenclature

Note that for purposes of brevity, the e*Way Intelligent Adapter for PeopleSoft Batch is frequently referred to as the PeopleSoft Batch e*Way, or simply the e*Way.

P.4 Online Use

This User's Guide is provided in Adobe Acrobat's Portable Document Format (PDF). As such, it can be printed out on any printer or viewed online. When viewing online, you can take advantage of the extensive hyperlinking imbedded in the document to navigate quickly throughout the Guide.

Hyperlinking is available in:

- The Table of Contents
- The Index
- Within the chapter text, indicated by **blue print**

Existence of a hyperlink *hotspot* is indicated when the hand cursor points to the text. Note that the hotspots in the Index are the *page numbers*, not the topics themselves. Returning to the spot you hyperlinked from is accomplished by right-clicking the mouse and selecting **Go To Previous View** on the resulting menu.

P.5 Writing Conventions

The writing conventions listed in this section are observed throughout this document.

Monospaced (Courier) Font

Computer code and text to be typed at the command line are set in Courier as shown below.

```
Configuration for BOB_Promotion
java -jar ValidationBuilder.jar
```

Variables within a command line are set in Courier italic as shown below.

```
stcregutl -rh host-name -un user-name -up password -sf
```

Bold Sans-serif Font

- User Input: Click **Apply** to save, or **OK** to save and close.
- File Names and Paths: In the **Open** field, type **D:\setup\setup.exe**.
- Parameter, Function, and Command Names: The default parameter **localhost** is normally only used for testing; the Monk function **iq-put** places an Event into an IQ.

P.6 Additional Documentation

- Many of the procedures included in this User's Guide are described in greater detail in the *e*Gate Integrator User's Guide*.
- For descriptions of the Batch e*Way, see the *Batch e*Way Intelligent Adapter User's Guide*.
- For descriptions of the ODBC e*Way, see the *e*Way Intelligent Adapter for ODBC User's Guide*.

Introduction

1.1 Overview

The PeopleSoft Batch e*Way uses selected features of the SeeBeyond Batch e*Way to handle batch data via FTP. It also can use selected features of the SeeBeyond ODBC e*Way (or Oracle e*Way) to provide direct access to the PeopleSoft Interface Tables for loading or extracting batch data. The following sections describe both of those e*Ways. For more detailed information, see [Chapter 5](#).

1.1.1 Batch e*Way

The Batch e*Way supports standard FTP according to RFC-959. The Batch e*Way supports the following commands:

APPE	NOOP	RNTO
CWD	PASS	SITE
DELE	QUIT	STOR
LIST	RETR	TYPE
MKD	RNFR	USER

1.1.2 ODBC e*Way

The ODBC e*Way uses a Monk extension library to issue SQL (Structured Query Language) statements. The library contains functions to access the database and generate SQL statements. SQL is the language used to communicate with the database server to access and manipulate data. By populating a database with the data flowing through an integration engine, all the information available to an integrated delivery network (IDN) is stored for evaluation. This allows the ODBC e*Way to operate independently of the underlying DBMS (database management system).

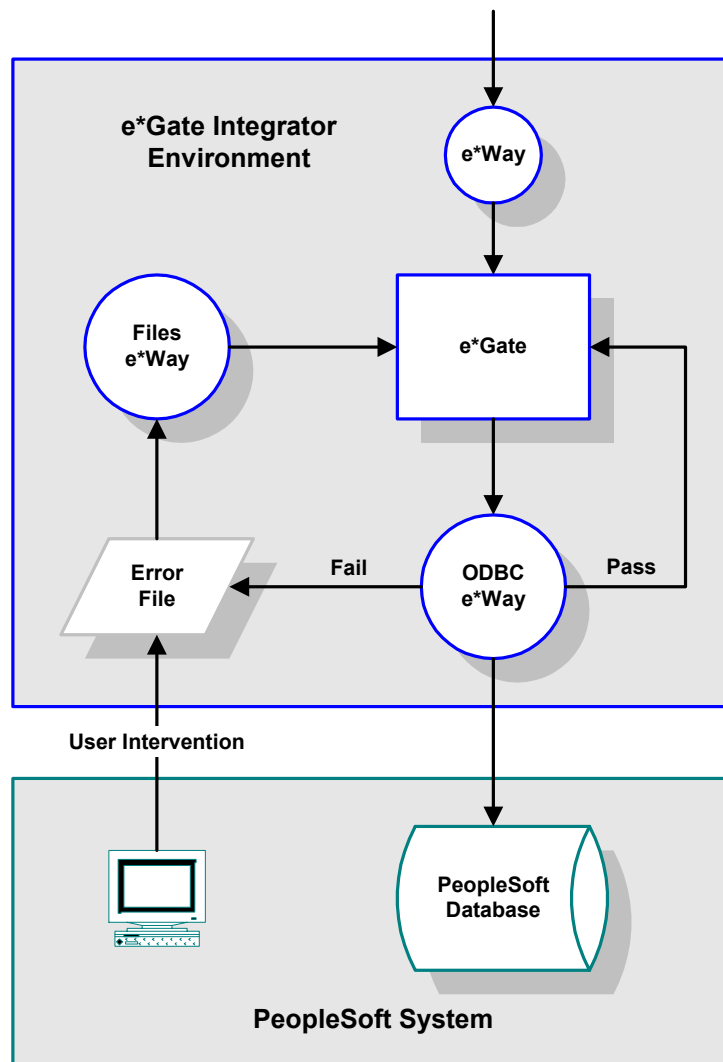
To access the database, you execute an SQL command, which is the American National Standards Institute (ANSI) standard language for operating upon relational databases. The language contains a large set of operators for defining and manipulating tables. SQL statements can be used to create, alter, and drop tables from a database.

1.1.3 e*Gate to PeopleSoft

A data file from the source application is passed to e*Gate for transformation, and then loaded directly into the PeopleSoft application tables using SQL commands from the ODBC e*Way. The main components of this PeopleSoft-inbound batch-mode interface include:

- A source-specific e*Way, to transfer a source data file.
- e*Gate Integrator, to route the data.
- An ODBC e*Way, to transform and load the data into the PeopleSoft system
- A file-handling e*Way, to process failed transactions.

Figure 1 e*Gate-to-PeopleSoft Interface Overview

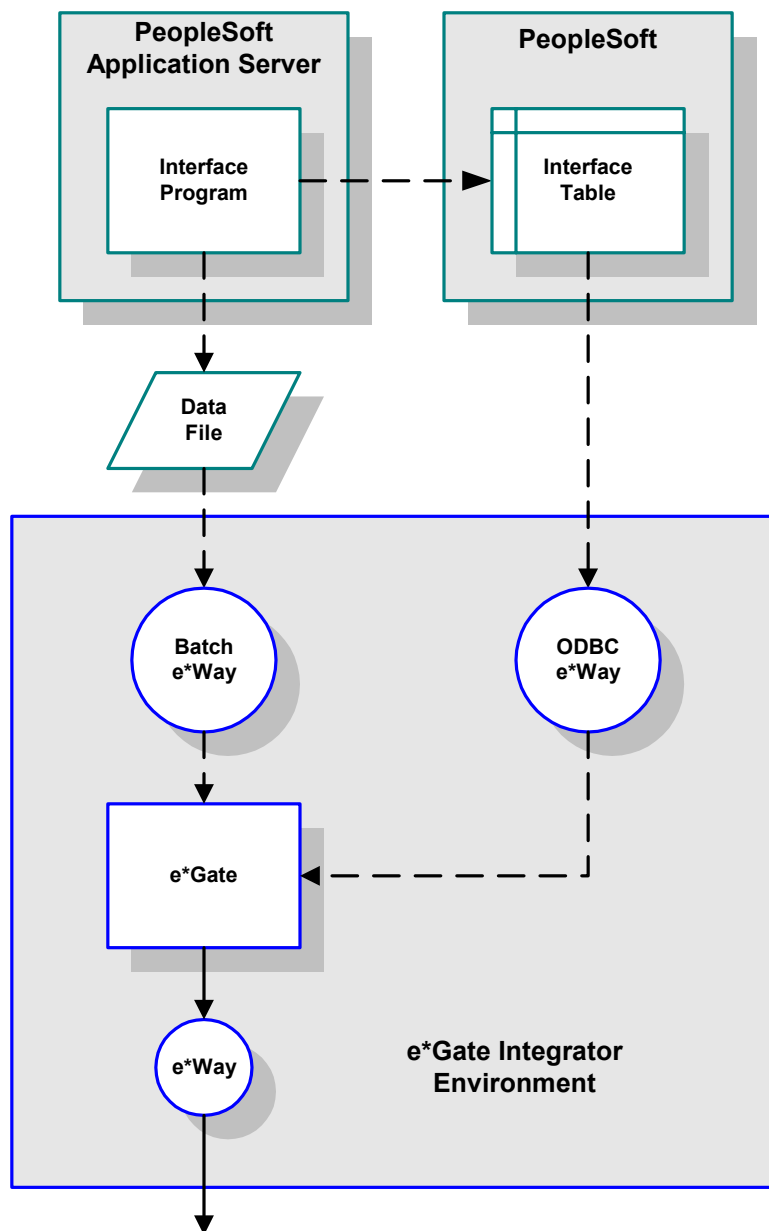


1.1.4 PeopleSoft to e*Gate

A PeopleSoft-outbound batch-mode interface involves the extraction of data from the PeopleSoft application tables, the processing and manipulation of that data, and the placement of the resulting data into an interface table or file. The data in the interface table or file is then sent to e*Gate for transformation and routing. The main components of an outbound batch interface include:

- An e*Way (either ODBC or Batch), to retrieve a data table or file.
- e*Gate, to transform and route the data.

Figure 2 PeopleSoft-to-e*Gate Interface Overview



The PeopleSoft-to-e*Gate interface was designed to be as flexible as possible. Most batch interfaces run on a scheduled basis, but this design also allows for an interface to be initiated by a remote function call (RFC) within the interface program. The design can accommodate interface data written to either an interface table or an interface file.

1.1.5 Components

The PeopleSoft Batch e*Way incorporates the following components:

- Executable file (installed with e*Gate Integrator)
 - ♦ **stcewgenericmonk.exe**
- Default configuration files
 - ♦ **batch.def**
 - ♦ **dart.def**
- Monk function scripts, discussed in [Chapter 6](#)
- Monk library files, discussed in [Chapter 7](#)

For a list of installed files, see [Chapter 2](#).

1.2 Supported Operating Systems

The e*Way Intelligent Adapter for PeopleSoft Batch is available on the following operating systems:

- Windows 2000 and Windows Server 2003
- Sun Solaris 8

Note: *The e*Gate Schema Designer runs only on Windows.*

Installation

This chapter describes the requirements and procedures for installing the e*Way software. Procedures for implementing a working system, incorporating instances of the e*Way, are described in [Chapter 3](#).

Note: Please read the *readme.txt* file located in the *addons\xxx* directory on the installation CD-ROM for important information regarding this installation.

2.1 System Requirements

To use the e*Way Intelligent Adapter for PeopleSoft Batch, you need the following:

- 1 An e*Gate Participating Host.
- 2 A TCP/IP network connection.
- 3 A connection to an FTP server.
- 4 Sufficient free disk space to accommodate e*Way files:
 - ♦ Approximately 1.2 MB on Windows systems
 - ♦ Approximately 5.1 MB on Solaris systems

Note: 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.

2.2 External System Requirements

- A *full version* of PeopleTools 7.5.3.
- PeopleSoft HRMS 7.5.
- A database client: i.e., Oracle7 Oracle8 or Oracle8i, Sybase (OpenClient), Informix (Client SDK), etc.
- An FTP server compliant with RFC-959.

Note: A user name and password granting appropriate access to the FTP server must be available for the e*Way's use.

2.2.1 SOCKS Support

SOCKS is a generic proxy protocol for TCP/IP-based networking applications. When an application client needs to connect to an application server, the client connects to a SOCKS proxy server. The proxy server connects to the application server on behalf of the client, and relays data between the client and the application server. For the application server, the proxy server is the client.

The Batch e*Way supports the SOCKS version 5 Authentication protocol. To enable SOCKS 5 support, the SOCKS server name and port number, as well as the user name and encrypted password, must be specified in the configuration file. Details of these configuration parameters are provided in the *Batch e*Way Intelligent Adapter User's Guide*.

2.2.2 OS/390 Support

OS/390 systems use the EBCDIC character set. As a consequence, ASCII-based systems cannot directly transport data to an EBCDIC-based system. ASCII to EBCDIC data conversion is necessary when data is sent from UNIX/Windows to OS/390. This data conversion should take place within the Collaboration.

To transport any EBCDIC data to an ASCII-based system (UNIX or Windows), you must first convert the data by using the **ebcdic->ascii** Monk function. Refer to the *Monk Developer's Reference* for details about that function.

See also the *Batch e*Way Intelligent Adapter User's Guide*, which describes options for data transfer modes to an FTP server.

2.3 Installing the e*Way

2.3.1 Windows Systems

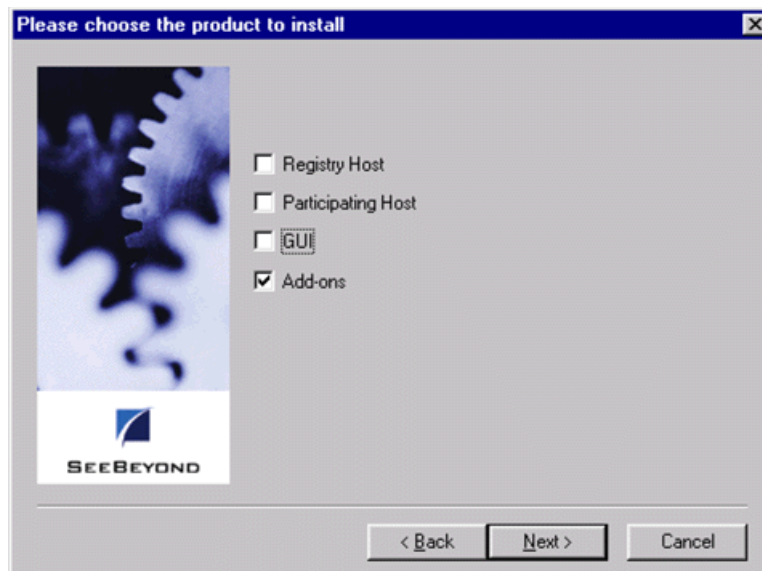
Installation Procedure

Note: *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond.*

To Install the e*Way on a Microsoft Windows System

- 1 Log in as an Administrator on the workstation on which you want to install the e*Way (you must have Administrator privileges to install this e*Way).
- 2 Exit all Windows programs and disable any anti-virus applications before running the setup program.
- 3 Insert the e*Way installation CD-ROM into the CD-ROM drive.
- 4 Launch the setup program.
 - A If the CD-ROM drive's Autorun feature is enabled, the setup program should launch automatically. Follow the on-screen instructions until the **Choose Product** dialog box appears (see Figure 3). Check **Add-ons**, then click **Next**.

Figure 3 Choose Product Dialog

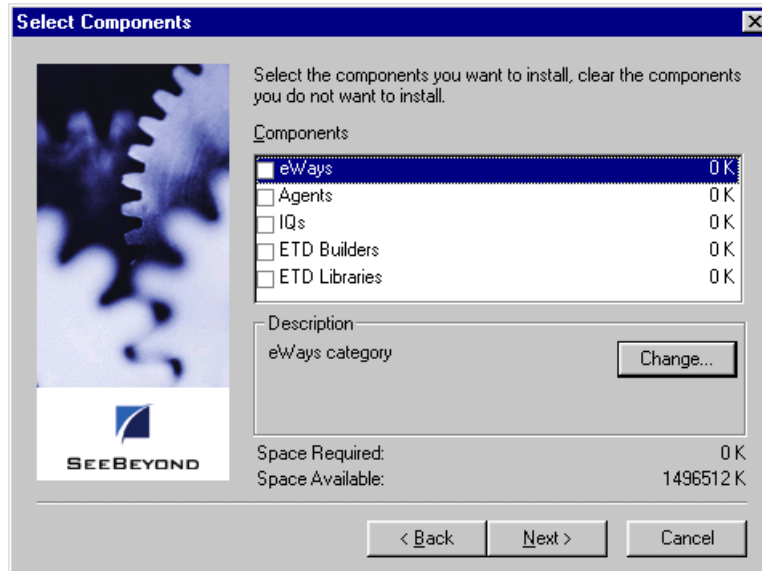


- B If the setup program does not launch automatically, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the following file on the CD-ROM drive (bypassing the **Choose Product** dialog):

```
setup\addons\setup.exe
```

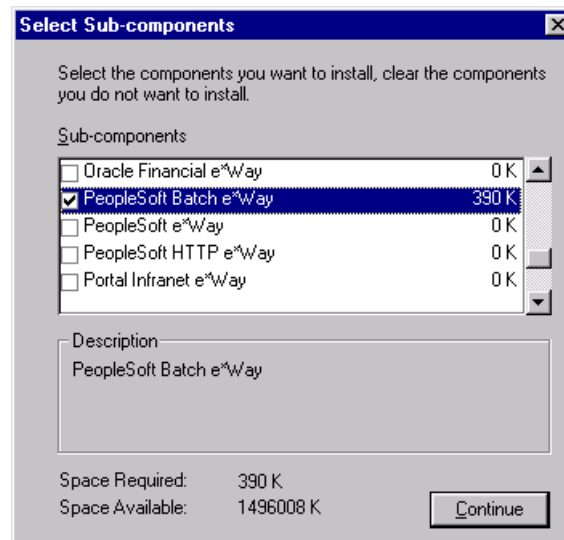

- 5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 4). Highlight—but do not check—**eWays** and then click **Change**.

Figure 4 Select Components Dialog



- 6 When the Select Sub-components dialog box appears (see Figure 5), check the **PeopleSoft Batch e*Way**.

Figure 5 Select e*Way Dialog



- 7 Click **Continue**, and the Select Components dialog box reappears.
- 8 Click **Next** and continue with the installation.

Subdirectories and Files

Note: *Installing the e*Way Intelligent Adapter for PeopleSoft Batch installs both the Batch e*Way and the ODBC e*Way. Both the Java and Monk versions of the Batch e*Way are installed; however, only the files used by the Monk version are used by this e*Way and, therefore, are listed in this section.*

By default, the InstallShield installer creates the following subdirectories and installs the following files within the \eGate\client tree on the Participating Host, and the \eGate\Server\registry\repository\default tree on the Registry Host.

Table 1 Participating Host & Registry Host

Subdirectories	Files
\bin\	stcodbctest.exe stcstruct.exe stc_dbapps.dll stc_dbmonkext.ctl stc_dbmonkext.dll stc_dbodbc.dll stc_ewftp.ctl stc_ewftp.dll stc_monkfilesys.dll
\configs\FtpHeuristics\	FtpHeuristics.cfg
\configs\stcewgenericmonk\	batch.def batchFtpRule.txt dart.def dartRule.txt ScFtp3.6To4.1Rule.txt
\etd\	dbwizard.ctl
\monk_library\	dart.gui
\monk_library\batch\ (continued on next page)	batch-ack.monk batch-dynamic-init.monk batch-dynamic-proc-out.monk batch-dynamic-send-to-egate.monk batch-exchange-data.monk batch-exchange-utils.monk batch-ext-connect.monk batch-ext-shutdown.monk batch-ext-verify.monk batch-fetch-files-from-remote.monk batch-fetch-named-files.monk batch-init.monk batch-nak.monk batch-persist.monk batch-post-transfer.monk batch-proc-out.monk batch-regular-init.monk batch-regular-proc-out.monk batch-send-path-file.monk

Table 1 Participating Host & Registry Host

Subdirectories	Files
\monk_library\batch\ (continued)	batch-shutdown-notify.monk batch-startup.monk batch-utils.monk batch-validate-params.monk file-ext-connect.monk file-ext-shutdown.monk file-ext-verify.monk file-fetch-path.monk file-fetch.monk file-init.monk file-remote-path-list.monk file-remote-post-transfer.monk file-rmt-list.monk file-rmt-post-transfer.monk file-send-path-file.monk file-send.monk file-startup.monk file-validate-params.monk ftp-connect.monk ftp-disconnect.monk ftp-ext-connect.monk ftp-ext-shutdown.monk ftp-ext-verify.monk ftp-fetch-path.monk ftp-fetch.monk ftp-init.monk ftp-pre-post-commands.monk ftp-remote-path-list.monk ftp-remote-post-transfer.monk ftp-rmt-list.monk ftp-rmt-post-transfer.monk ftp-send-path-file.monk ftp-send.monk ftp-startup.monk ftp-validate-params.monk local-post-transfer.monk
\monk_library\common\	batch_eway_data.dtd batch_eway_data.ssc batch_eway_error.dtd batch_eway_error.ssc batch_eway_order.dtd batch_eway_order.ssc

Table 1 Participating Host & Registry Host

Subdirectories	Files
\monk_library\dart\	db-sanitize-symbol.monk db-stdver-eway-funcs.monk db-struct-bulk-insert.monk db-struct-call.monk db-struct-execute.monk db-struct-fetch.monk db-struct-insert.monk db-struct-select.monk db-struct-update.monk db2msg-display.monk db2msg.ssc db_bind.monk odbcmsg-display.monk odbcmsg.ssc oramsg-display.monk oramsg.ssc sybmsg-display.monk sybmsg.ssc
\stcgui\ctls\	guidart.ctl

By default, the InstallShield installer also installs the following files within the \eGate\Server\registry\repository\default tree on the Registry Host.

Table 2 Registry Host Only

Subdirectories	Files
\	stcewbatch.ctl stcewodbc.ctl

Environment Configuration

No changes are required to the Participating Host's operating environment to support this e*Way.

2.3.2 UNIX Systems

Installation Procedure

Note: You are not required to have root privileges to install this e*Way. Log on under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privilege to create files in the e*Gate directory tree.

To install the e*Way on a UNIX system

- 1 Log onto the workstation containing the CD-ROM drive and, if necessary, mount the drive.
- 2 Insert the e*Way installation CD-ROM into the CD-ROM drive.
- 3 At the shell prompt, type
`cd /cdrom`
- 4 Start the installation script by typing:
`setup.sh`
- 5 A menu appears, with several options. Select the **Install e*Way** option, and follow any additional on-screen instructions.

Note: The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond. Note also that **no spaces** should appear in the installation path name.

Subdirectories and Files

Note: Installing the e*Way Intelligent Adapter for PeopleSoft Batch installs both the Batch e*Way and the ODBC e*Way. Both the Java and Monk versions of the Batch e*Way are installed; however, only the files used by the Monk version are used by this e*Way and, therefore, are listed in this section.

The preceding installation procedure installs the following files only within the /eGate/Server/registry/repository/default tree on the Registry Host.

Table 3 Registry Host Only

Subdirectories	Files
/	stcewbatch.ctl stcewodbc.ctl

The preceding installation procedure also creates the following subdirectories and installs the following files within the /eGate/client tree on the Participating Host, and the /eGate/Server/registry/repository/default tree on the Registry Host.

Table 4 Participating Host & Registry Host

Subdirectories	Files
/bin/	stcodbctest.exe stcstruct.exe stc_dbapps.dll stc_dbmonkext.ctl stc_dbmonkext.dll stc_dbodbc.dll stc_ewftp.ctl stc_ewftp.dll stc_monkfilesys.dll
/configs/FtpHeuristics/	FtpHeuristics.cfg
/configs/stcewgenericmonk/	batch.def batchFtpRule.txt dart.def dartRule.txt ScFtp3.6To4.1Rule.txt
/etd/	dbwizard.ctl
/monk_library/	dart.gui
/monk_library/batch/	batch-ack.monk batch-dynamic-init.monk batch-dynamic-proc-out.monk batch-dynamic-send-to-egate.monk batch-exchange-data.monk batch-exchange-utils.monk batch-ext-connect.monk batch-ext-shutdown.monk batch-ext-verify.monk batch-fetch-files-from-remote.monk batch-fetch-named-files.monk batch-init.monk batch-nak.monk batch-persist.monk batch-post-transfer.monk batch-proc-out.monk batch-regular-init.monk batch-regular-proc-out.monk batch-send-path-file.monk batch-shutdown-notify.monk batch-startup.monk batch-utils.monk batch-validate-params.monk file-ext-connect.monk file-ext-shutdown.monk file-ext-verify.monk file-fetch.monk file-fetch-path.monk file-init.monk file-remote-path-list.monk
(continued on next page)	

Table 4 Participating Host & Registry Host

Subdirectories	Files
/monk_library/batch/ (continued)	file-remote-post-transfer.monk file-rmt-list.monk file-rmt-post-transfer.monk file-send-path-file.monk file-send.monk file-startup.monk file-validate-params.monk ftp-connect.monk ftp-disconnect.monk ftp-ext-connect.monk ftp-ext-shutdown.monk ftp-ext-verify.monk ftp-fetch-path.monk ftp-fetch.monk ftp-init.monk ftp-pre-post-commands.monk ftp-remote-path-list.monk ftp-remote-post-transfer.monk ftp-rmt-list.monk ftp-rmt-post-transfer.monk ftp-send-path-file.monk ftp-send.monk ftp-startup.monk ftp-validate-params.monk local-post-transfer.monk
/monk_library/common/	batch_eway_data.dtd batch_eway_data.ssc batch_eway_error.dtd batch_eway_error.ssc batch_eway_order.dtd batch_eway_order.ssc
/monk_library/dart/	db-sanitize-symbol.monk db-stdver-eway-funcs.monk db-struct-bulk-insert.monk db-struct-call.monk db-struct-execute.monk db-struct-fetch.monk db-struct-insert.monk db-struct-select.monk db-struct-update.monk db2msg-display.monk db2msg.ssc db_bind.monk odbcmmsg-display.monk odbcmmsg.ssc oramsg-display.monk oramsg.ssc sybmsg-display.monk sybmsg.ssc

Table 4 Participating Host & Registry Host

Subdirectories	Files
/stcgui/ctls/	guidart.ctl

Environment Configuration

No changes are required to the Participating Host's operating environment to support this e*Way.

2.4 Optional Example Files

The installation CD contains two sample schema, **IB_psoft** and **OB_psoft**, located in the **samples\ewpssoft** directory. To use these schemas, you must load them onto your system using the following procedure. See **Sample Schema** on page 35 for descriptions of the sample schema and instructions regarding its use.

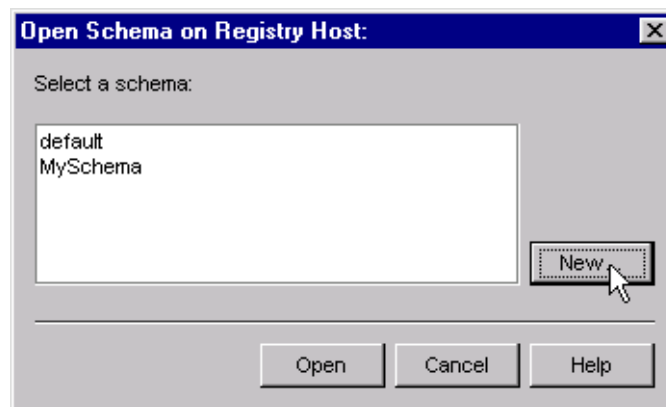
Note: *The PeopleSoft Batch e*Way must be properly installed on your system before you can run the sample schema.*

2.4.1 Installation Procedure

To load a sample schema

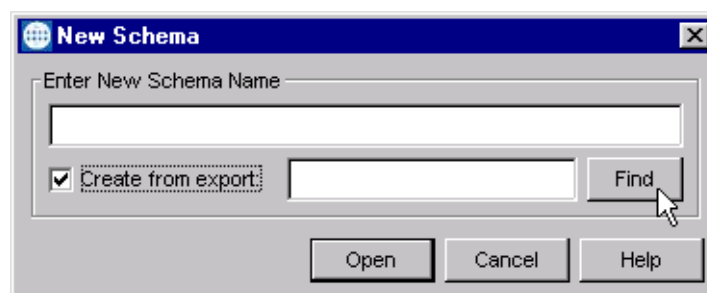
- 1 Invoke the **Open Schema** dialog box and select **New** (see Figure 6).

Figure 6 Open Schema Dialog



- 2 Type the name you want to give to the schema (for example, **IB.Sample**)
- 3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 7).

Figure 7 New Schema Dialog



- 4 Select the desired archive file (*.zip) and click **Open**.

Note: The schema installs with the host name **localhost** and control broker name **localhost_cb**. If you want to assign your own names, copy the file ***.zip** to a local directory and extract the files. Using a text editor, edit the file ***.exp**, replacing all instances of the name **localhost** with your desired name. Add the edited **.exp** file back into the **.zip** file.

2.4.2 Subdirectories and Files

The preceding procedure creates the following subdirectories and installs the following files within the `\eGate\Server\registry\repository\<SchemaName>` tree on the Registry Host, where `<SchemaName>` is the name you have assigned to the schema in step 2.

Table 5 Subdirectories and Files - IB_psoft

Subdirectories	Files
\	IB_psoft.ctl
\runtime\configs\stcewfile\	dartfeeder.cfg dartfeeder.sc
\runtime\configs\stcewgenericmonk\	Dart.cfg Dart.sc
\runtime\data\	IB_DART.dat
\runtime\monk_scripts\common\	IB_ERROR_HANDLER.tsc RIDE_IB.ssc RIDE_IB_DART.dsc RIDE_IB_DART.ssc

Table 6 Subdirectories and Files - OB_psoft

Subdirectories	Files
\	OB_psoft.ctl
\runtime\configs\stcewfile\	eater.cfg eater.sc
\runtime\configs\stcewgenericmonk\	dartpoll.cfg dartpoll.sc
\runtime\data\	Stc_ride.dat Stc_ride.dms
\runtime\monk_scripts\common\	OB_DART.dsc OB_DART.ssc

System Implementation

In this chapter we summarize the procedures required for implementing a working system incorporating the e*Way Intelligent Adapter for PeopleSoft Batch. Please see the *e*Gate Integrator User's Guide* for additional information.

3.1 Overview

This e*Way provides a specialized transport component for incorporation into an operational Schema. The schema also contains Collaborations, linking different data or Event types, and Intelligent Queues. Typically, other e*Way types also are used as components of the Schema.

Topics included in this chapter include:

[Creating a Schema](#) on page 29

[Creating Event Types](#) on page 30

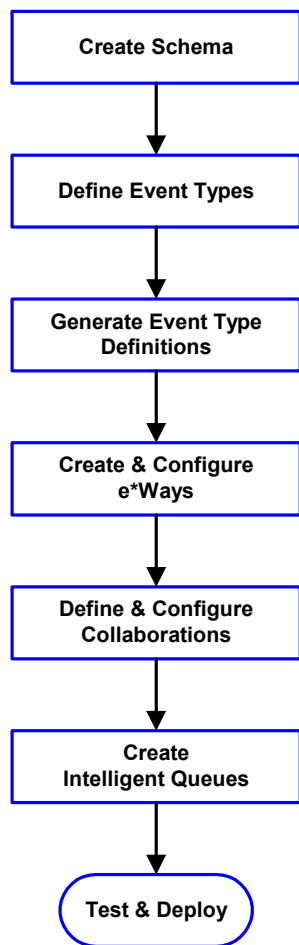
[Generating Event Type Definitions](#) on page 31

[Defining Collaborations](#) on page 34

[Creating Intelligent Queues](#) on page 35

[Sample Schema](#) on page 35

3.1.1 Implementation Sequence



- 1 The first step is to create a new Schema—the subsequent steps apply only to this Schema (see [Creating a Schema](#) on page 29).
- 2 The second step is to define the Event Types you are transporting and processing within the Schema (see [Creating Event Types](#) on page 30).
- 3 Third, you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) derived from the applicable Business Rules (see [Generating Event Type Definitions](#) on page 31).
- 4 The fourth step is to create and configure the required e*Ways (see [Chapter](#)).
- 5 Next is to define and configure the Collaborations linking the Event Types from step 2 (see [Defining Collaborations](#) on page 34).
- 6 Now you need to create Intelligent Queues to hold published Events (see [Creating Intelligent Queues](#) on page 35).
- 7 Finally, you must test your Schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

3.1.2 Viewing e*Gate Components

Use the Navigator and Editor panes of the e*Gate Schema Designer to view the various e*Gate components. Note that you may only view components of a single schema at one time, and that all operations apply only to the current schema. All procedures in this chapter should be performed while displaying the **Components** Navigator pane. See the *e*Gate Integrator User's Guide* for a detailed description of the features and use of the Schema Designer.

3.2 Creating a Schema

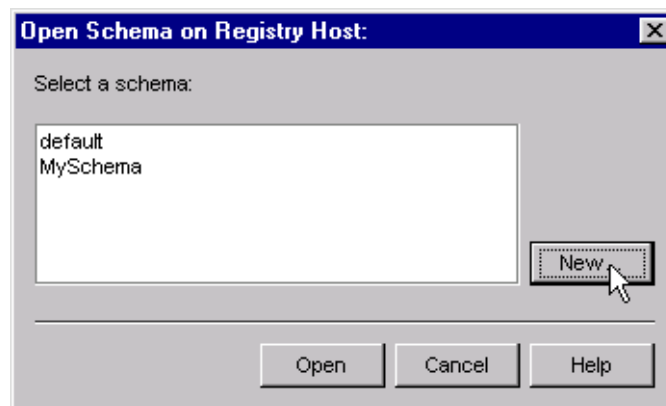
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

Because all setup and configuration operations take place within an e*Gate schema, a new schema must be created, or an existing one must be started before using the system. Schemas store all their configuration parameters in the e*Gate Registry.

To select or create a schema

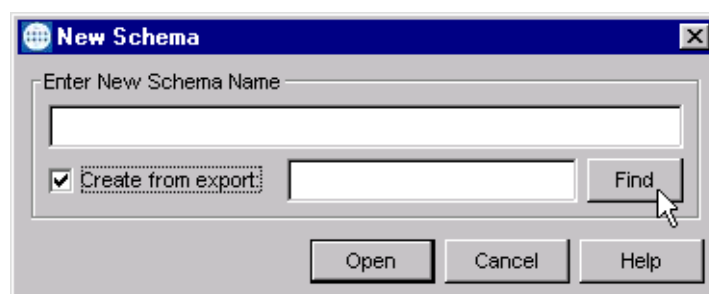
- 1 Invoke the **Open Schema** dialog box and **Open** an existing schema or click **New** to create a new schema.

Figure 8 Open Schema Dialog



- 2 Clicking **New** invokes the **New Schema** dialog box (Figure 9).

Figure 9 New Schema Dialog




- 3 Enter a new schema name and click **Open**.
- 4 The e*Gate Schema Designer then opens under your new schema name.
- 5 From the **Options** menu, click on **Default Editor** and select **Monk**.
- 6 Select the **Components** tab, found at the bottom of the Navigator pane of the e*Gate Schema Designer window.
- 7 You are now ready to begin creating the necessary components for this new schema.

3.3 Creating Event Types

Within e*Gate, messages and/or packages of data are defined as Events. Each Event must be categorized into a specific Event Type within the schema.

To define the Event Types

- 1 In the e*Gate Schema Designer's Navigator pane, select the **Event Types** folder.
- 2 On the Palette, click the **New Event Type** button .
- 3 In the **New Event Type Component** box, enter the name for the input Event Type and click **Apply**. Use this method to create all required Event Types, for example:
 - ◆ **InboundEvent**
 - ◆ **ValidEvent**
 - ◆ **InvalidEvent**
- 4 After you have created the final Event Type, click **OK**.


3.4 Generating Event Type Definitions

3.4.1 Using the ETD Editor's Build Tool

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.

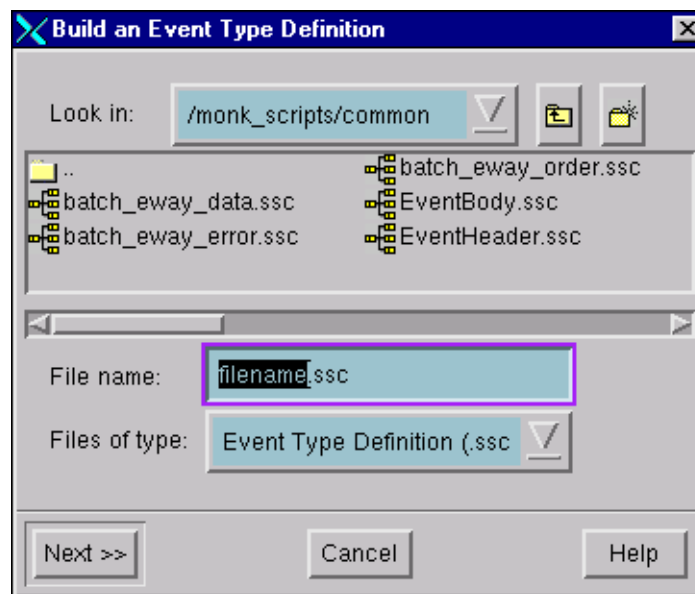
Note: Be sure to set the Default Editor to **Monk**, from the **Options** menu in the e*Gate Schema Designer.

To create an Event Type Definition using the Build tool

- 1 Launch the ETD Editor by clicking  in the e*Gate Schema Designer tool bar.
- 2 On the ETD Editor's tool bar, click **Build**.

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

Figure 10 Build Event Type Definition Dialog



- 3 In the *File name* box, type the name of the ETD file you want to build.

Note: The Editor automatically supplies the *.ssc* extension.

- 4 Click **Next**.
- 5 Under *Build From*, select **Library Converter**.
- 6 Under *Select a Library Converter*, select **Database Converter**.

- 7 In the *Additional Command Line Arguments* box, type any additional arguments, if desired.
- 8 Click **Finish**. This builds the ETD file.

3.4.2 Assigning ETDs to Event Types

After you have created the e*Gate system's ETD files, you can assign them to Event Types you have already created.

To assign ETDs to Event Types


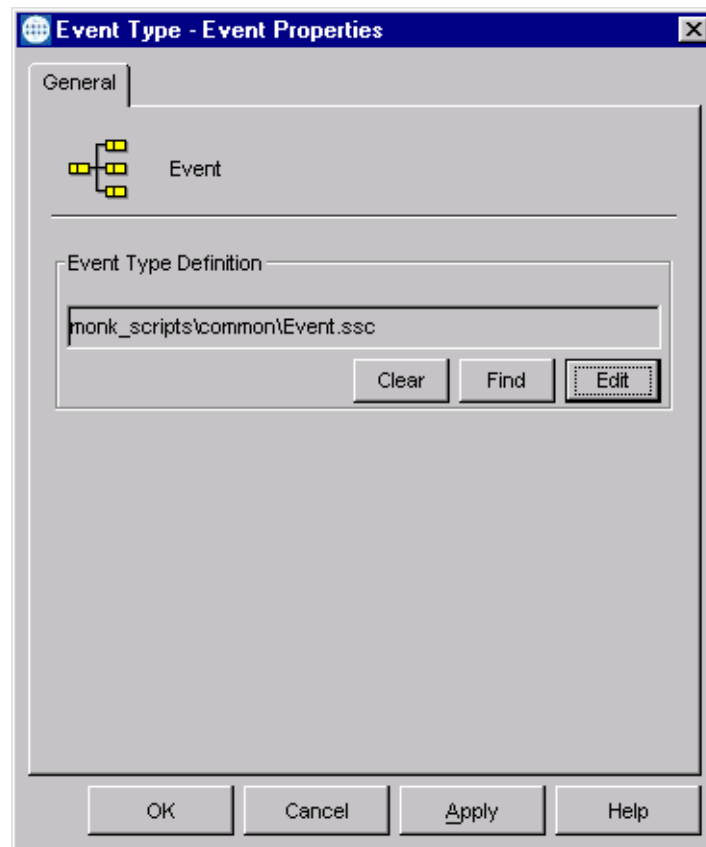
- 1 In the Schema Designer window, select the **Event Types** folder in the Navigator/Components pane.
- 2 In the Editor pane, select one of the Event Types you created.
- 3 Right-click on the Event Type and select **Properties** (or click  in the toolbar). The Event Type Properties dialog box appears. See Figure 11.

Figure 11 Event Type Properties Dialog Box



- 4 Under Event Type Definition, click **Find**, and the Event Type Definition Selection dialog box appears (it is similar to the Windows Open dialog box).
- 5 Open the **monk_scripts** folder, then select the desired file name (.ssc). It is usually found in the **common** sub-folder, but may be in a sub-folder specific to the e*Way.
- 6 Click **Select**. The file populates the Event Type Definition field.

- 7 To save any work in the properties dialog box, click **Apply** to enter it into the system.
- 8 When finished assigning ETDs to Event Types, click **OK** to close the properties dialog box and apply all the properties.

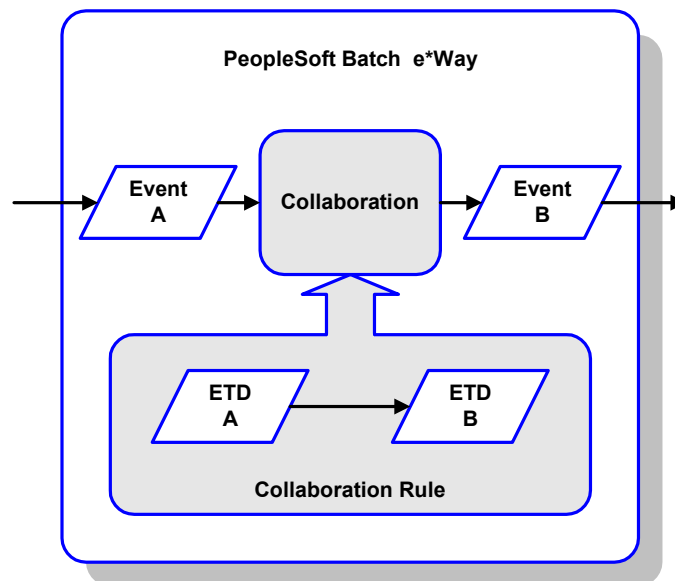
Each Event Type is now associated with the specified Event Type Definition.

3.5 Defining Collaborations

After you have created the required Event Type Definitions, you must define a Collaboration to transform the incoming Event into the desired outgoing Event.

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

Figure 12 Collaborations



The Collaboration is driven by a Collaboration Rule script, which defines the relationship between the incoming and outgoing ETDs. You can use an existing Collaboration Rule script, or use the Monk programming language to write a new Collaboration Rule script. Once you have written and successfully tested a script, you can then add it to the system’s run-time operation.

Collaborations are defined using the e*Gate Monk Collaboration Rules Editor. See the *e*Gate Integrator User’s Guide* for instructions on using this Editor. The file extension for Monk Collaboration Rules is **.tsc**.

3.6 Creating Intelligent Queues

The final step is to create and associate an IQ for the PeopleSoft Batch e*Way. IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another. IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database). See the *e*Gate Integrator User's Guide* for complete information on queuing options and procedures for creating IQs.

3.7 Sample Schema

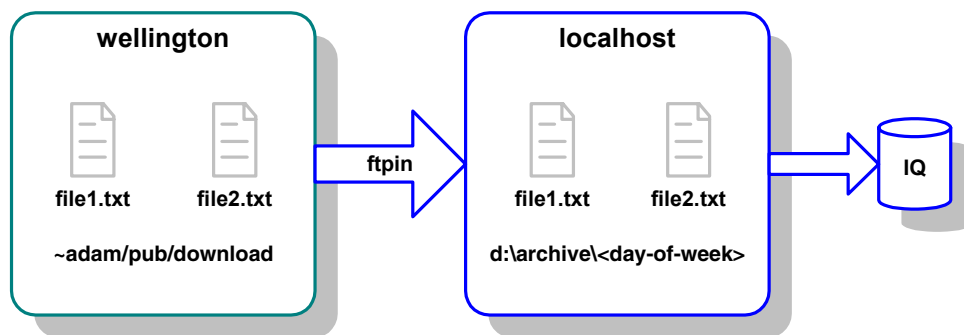
The e*Gate CD includes two sample schemas, located in the **samples\ewpsoft** directory, that illustrate both the inbound and outbound modes of operation. See [Optional Example Files](#) on page 25 for information on sample schema installation.

3.7.1 Subscribing to an External System

In this example, the Batch e*Way fetches two files from the remote UNIX machine **wellington** every 24 hours, using the FTP protocol. These files are stored in the home directory of user **adam**, under the subdirectory **pub/download**.

Figure 13 below shows a diagram of this setup.

Figure 13 Subscribe-to-external-system Setup



This setup has the following additional characteristics:

- The names of the two files are **file1.txt** and **file2.txt**. No other files are required.
- The two files contain multiple records delimited by a new line (\n) character.
- After retrieving the files from the remote system, the Batch e*Way deletes the remote copy.
- The last seven day's worth of files on the local system are kept.

The Table 7 lists the most critical parameters and the settings required to achieve the setup described previously.

Table 7 Parameters for the Input Example

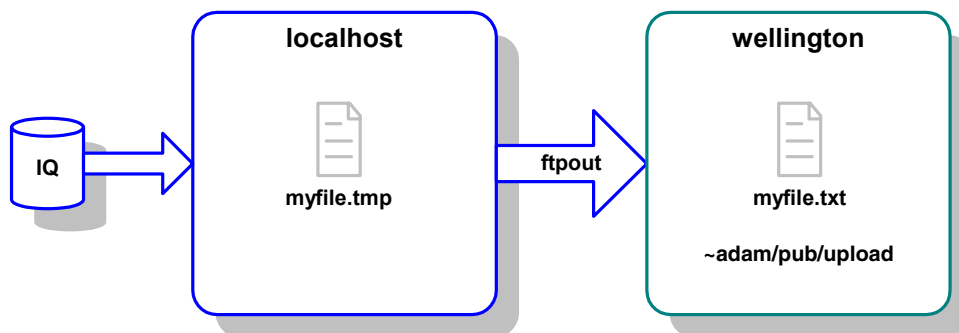
Section	Parameter	Value
Communication Setup	Start Exchange Data Schedule	Repeatedly, every 24 hours
External Host Setup	Host Type	UNIX
	External Host Name	wellington
	User Name	adam
	Encrypted Password	*****
	File Transfer Method	FTP
Subscribe To External	Remote Directory Name	pub/download
	Remote File Regexp	^file[12].txt\$
	Record Type	Delimited
	Record Delimiter	\n
	Delimiter on Last Record	Yes
	Remote Command After Transfer	delete
	Local Command After Transfer	archive
	Local Rename or Archive Name	d:\archive/%a

3.7.2 Publishing to an External System

In this example, the Batch e*Way sends a file containing new line (\n) delimited messages to the remote UNIX machine **wellington**, using the FTP protocol. The file is created in the subdirectory **pub/upload**, under the user **adam**.

Figure 14 below shows a diagram of this setup.

Figure 14 Publish-to-external-system Setup



This file is sent once every hour under the name **myfile.tmp**, and is renamed **myfile.txt** after it arrives. This technique can be used if there is a process on the remote machine

watching for a file to be created, but we want to make sure that it does not see the file until it is there in its entirety.

A copy of the file on the local system is not required and is deleted.

The Table 8 below lists the most critical parameters and the settings required to achieve the setup described previously.

Table 8 Parameters for the Output Example

Section	Parameter	Value
Communication Setup	Start Exchange Data Schedule	Repeatedly, every 1 hour
External Host Setup	Host Type	UNIX
	External Host Name	wellington
	User Name	adam
	Encrypted Password	*****
	File Transfer Method	FTP
Publish To External	Remote Directory Name	pub/upload
	Remote File Name	myfile.tmp
	Append or Overwrite when Transferring Files	Overwrite
	Record Type	Delimited
	Record Delimiter	\n
	Delimiter on Last Record	Yes
	Remote Command After Transfer	rename
	Local Command After Transfer	delete
	Remote Rename or Archive Name	myfile.txt

Setup Procedures

This chapter summarizes the setup procedures for the PeopleSoft Batch e*Way.

4.1 Overview

After creating a schema, you must instantiate and configure the PeopleSoft Batch e*Way to operate within the schema. A wide range of setup options allow the e*Way to conform to your system's operational characteristics and your facility's operating procedures.

The topics discussed in this chapter include the following:

Setting Up the e*Way

[Creating the e*Way](#) on page 39

[Modifying e*Way Properties](#) on page 40

[Configuring the e*Way](#) on page 41

[Changing the User Name](#) on page 45

[Setting Startup Options or Schedules](#) on page 45

[Activating or Modifying Logging Options](#) on page 47

[Activating or Modifying Monitoring Thresholds](#) on page 48

Troubleshooting the e*Way

[Configuration Problems](#) on page 49

[System-related Problems](#) on page 50

4.2 Setting Up the e*Way

Note: The e*Gate Schema Designer GUI runs only on the Windows operating system.

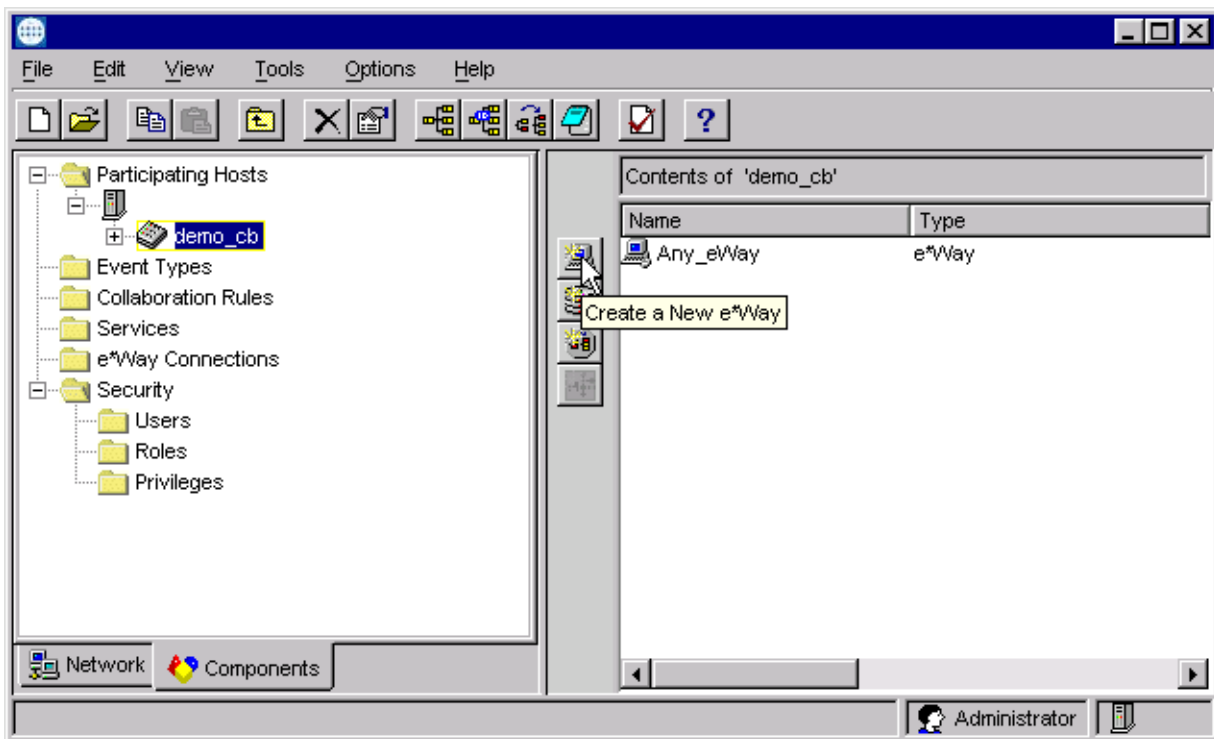
4.2.1 Creating the e*Way

The first step in implementing an e*Way is to define the e*Way component using the e*Gate Schema Designer.

To create an e*Way

- 1 Open the schema in which the e*Way is to operate.
- 2 Select the e*Gate Schema Designer Navigator's **Components** tab.
- 3 Open the host on which you want to create the e*Way.
- 4 Select the Control Broker you want to manage the new e*Way.

Figure 15 e*Gate Schema Designer Window (Components View)



- 5 On the Palette, click **Create a New e*Way**.
- 6 Enter the name of the new e*Way, then click **OK**.
- 7 All further actions are performed in the e*Gate Schema Designer Navigator's **Components** tab.

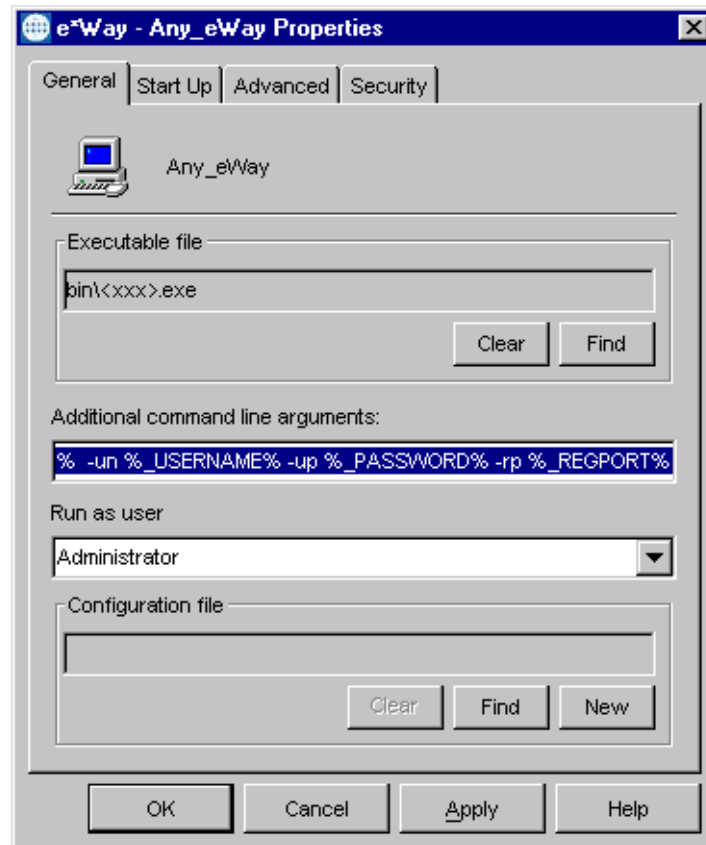
4.2.2 Modifying e*Way Properties

To modify any e*Way properties

- 1 Right-click on the desired e*Way and select **Properties** to edit the e*Way's properties. The properties dialog opens to the **General** tab (shown in Figure 16).

Note: The executable file is **stcewgenericmonk.exe**.

Figure 16 e*Way Properties (General Tab)



- 2 Make the desired modifications, then click **OK**.

4.2.3 Configuring the e*Way

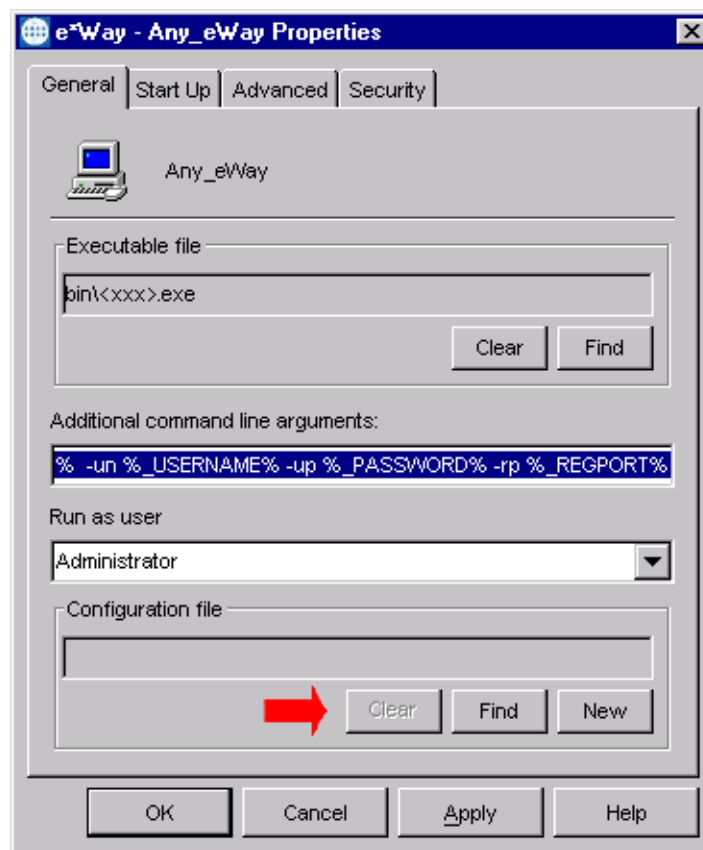
The e*Way's default configuration parameters are stored in an ASCII text file with a .def extension. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (.cfg) file.

To change e*Way configuration parameters

- 1 In the e*Gate Schema Designer's Component editor, select the e*Way you want to configure and display its properties.

Note: The default configuration files are **batch.def** and **dart.def**.

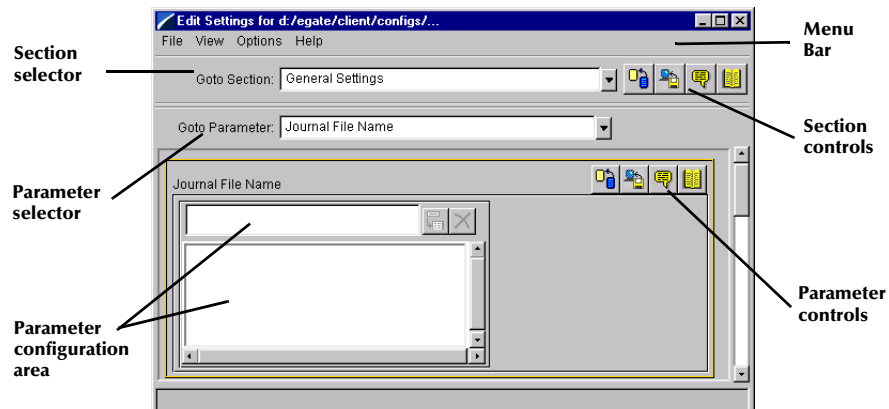
Figure 17 e*Way Properties - General Tab



- 2 Under **Configuration File**, click **New** to create a new file or **Find** to select an existing configuration file. If you select an existing file, an **Edit** button appears. Click this button to edit the currently selected file.
- 3 You are now in the e*Way Configuration Editor.

4.2.4 Using the e*Way Editor

Figure 18 The e*Way Configuration Editor







The e*Way Editor controls fall into one of six categories:

- The **Menu bar** allows access to basic operations (e.g., saving the configuration file, viewing a summary of all parameter settings, and launching the Help system)
- The **Section selector** at the top of the Editor window enables you to select the category of the parameters you wish to edit
- **Section controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected section
- The **Parameter selector** allows you to jump to a specific parameter within the section, rather than scrolling
- **Parameter controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected parameter
- **Parameter configuration controls** enable you to set the e*Way's various operating parameters

Section and Parameter Controls

The section and parameter controls are shown in Table 9 below.

Table 9 Parameter and Section Controls

Button	Name	Function
	Restore Default	Restores default values
	Restore Value	Restores saved values
	Tips	Displays tips
	User Notes	Enters user notes



Note: The section controls affect all parameters in the selected section, whereas the parameter controls affect only the selected parameter.

Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Option buttons
- Selection lists, which have controls as described in Table 10

Table 10 Selection List Controls

Button	Name	Function
	Add to List	Adds the value in the text box to the list of available values.
	Delete Items	Displays a "delete items" dialog box, used to delete items from the list.

Command-line Configuration

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.

Getting Help

To launch the e*Way Editor's Help system

From the **Help** menu, select **Help** topics.

To display tips regarding the general operation of the e*Way

From the **File** menu, select **Tips**.

To display tips regarding the selected Configuration Section

In the **Section Control** group, click .

To display tips regarding the selected Configuration Parameter

In the **Parameter Control** group, click .

Note: *“Tips” are displayed and managed separately from the online Help system. You cannot search for Tips within the Help system, or view Help system topics by requesting Tips.*

For detailed descriptions and procedures for using the e*Way Configuration Editor, see the *e*Gate Integrator User's Guide*.

4.2.5 Changing the User Name

Like all e*Gate executable components, e*Ways run under an e*Gate user name. By default, all e*Ways run under the **Administrator** user name. You can change this if your site's security procedures so require.

To change the user name

- 1 Display the e*Way's properties dialog.
- 2 On the **General** tab, use the **Run as user** list to select the e*Gate user under whose name this component is to run.

See the *e*Gate Integrator System Administration and Operations Guide* for more information on the e*Gate security system.

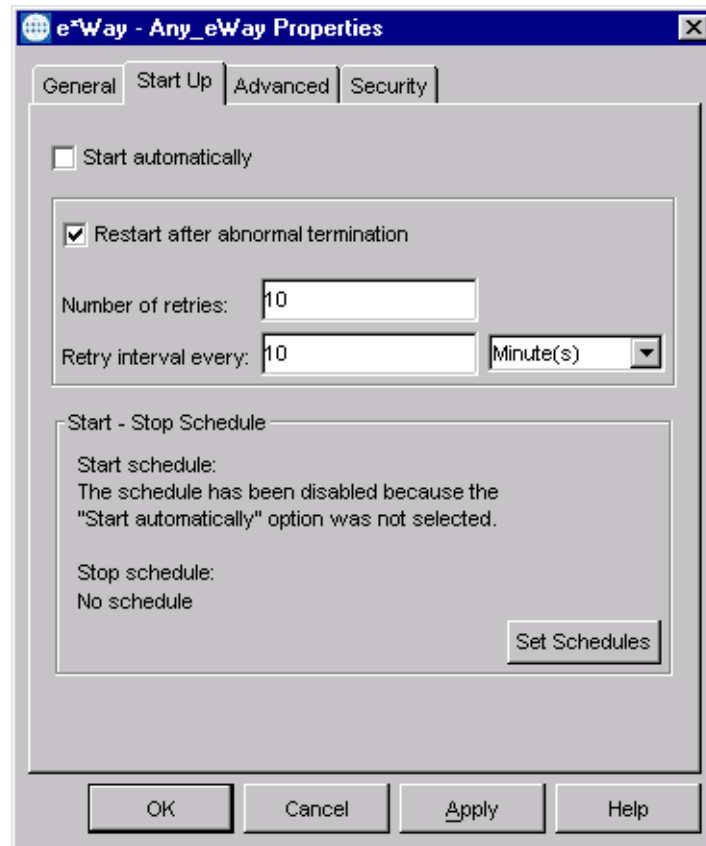
4.2.6 Setting Startup Options or Schedules

SeeBeyond e*Ways can be started or stopped by any of the following methods:

- The Control Broker can start the e*Way automatically whenever the Control Broker starts.
- The Control Broker can start the e*Way automatically whenever it detects that the e*Way terminated execution abnormally.
- The Control Broker can start or stop the e*Way on a schedule that you specify.
- Users can start or stop the e*Way manually using an interactive monitor.

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see Figure 19). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 19 e*Way Properties (Start-Up Tab)



To set the e*Way's startup properties

- 1 Display the e*Way's properties dialog.
- 2 Select the **Start Up** tab.
- 3 To have the e*Way start automatically when the Control Broker starts, select the **Start automatically** check box.
- 4 To have the e*Way start manually, clear the **Start automatically** check box.
- 5 To have the e*Way restart automatically after an abnormal termination:
 - A Select **Restart after abnormal termination**.
 - B Set the desired number of retries and retry interval.
- 6 To prevent the e*Way from restarting automatically after an abnormal termination, clear the **Restart after abnormal termination** check box.
- 7 Click **OK**.

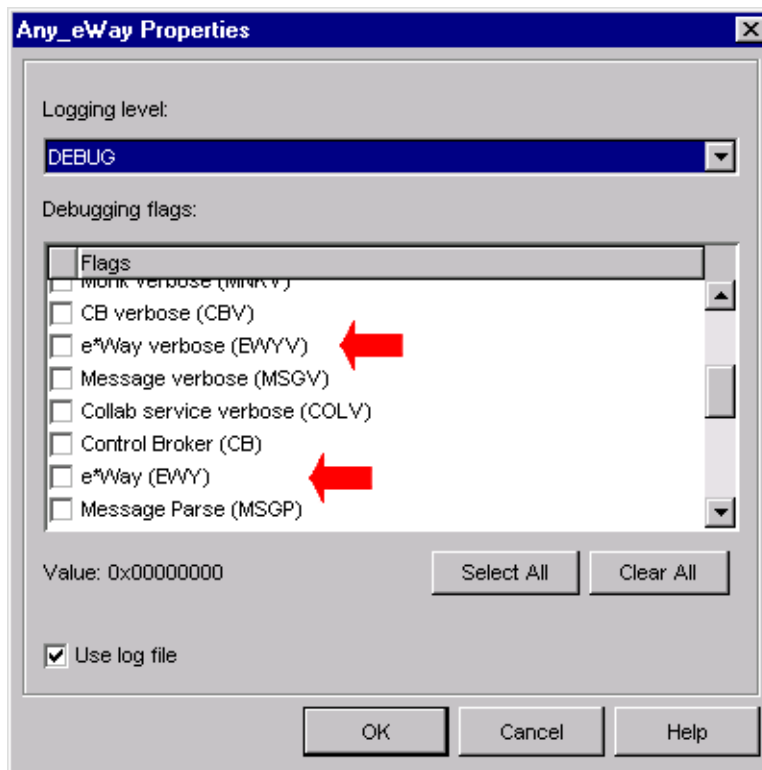
4.2.7 Activating or Modifying Logging Options

Logging options enable you to troubleshoot problems with the e*Way and other e*Gate components.

To set the e*Way debug level and flag

- 1 Display the e*Way's Properties dialog.
- 2 Select the **Advanced** tab.
- 3 Click **Log**. The dialog window appears (see Figure 20).

Figure 20 e*Way Properties (Advanced Tab - Log Option)



- 4 Select **DEBUG** for the **Logging level**.
- 5 Select either **e*Way (EWY)** or **e*Way Verbose (EWYV)** for the **Debugging flag**. Note that the latter has a significant negative impact on system performance.
- 6 Click **OK**.

The other options apply to other e*Gate components and are activated in the same manner. See the *e*Gate Integrator Alert and Log File Reference* for additional information concerning log files, logging options, logging levels, and debug flags.

4.2.8 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way sends a Monitoring Event to the Control Broker, which is then routed to the Schema Manager and any other configured destinations.

- 1 Display the e*Way's properties dialog.
- 2 Select the **Advanced** tab.
- 3 Click **Thresholds**.
- 4 Select the desired threshold options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference* for more information concerning threshold monitoring, routing specific notifications to specific recipients, or for general information about e*Gate's monitoring and notification system.

4.3 Troubleshooting the e*Way

In the initial stages of developing your e*Gate Integrator system administration system, most problems with e*Ways can be traced to configuration.

4.3.1 Configuration Problems

In the Schema Designer

- Does the e*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration Rule script employed by this e*Way's Collaborations?
- Do those Collaborations subscribe to and publish Events appropriately?
- Are all the components that *feed* this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e*Way *feeds* properly configured, and are they subscribing to the appropriate Events correctly?

In the e*Way Editor

- Check that all configuration options are set appropriately.
- Check that all settings you changed are set correctly.
- Check all required changes to ensure they have not been overlooked.
- Check the defaults to ensure they are acceptable for your installation.

On the e*Way's Participating Host

- Check that the Participating Host is operating properly, and that it has sufficient disk space to hold the IQ data that this e*Way's Collaborations publish.
- Check that the *path* environment variable includes the location of the PeopleSoft Batch dynamically-loaded libraries. The name of this variable on the different operating systems is:
 - ♦ PATH (Windows)
 - ♦ LD_LIBRARY_PATH (Solaris)

In the PeopleSoft Application

- Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately.

4.3.2 System-related Problems

- Check that the connection between the external application and the e*Way is functioning appropriately.
- Once the e*Way is up and running properly, operational problems can be due to:
 - ♦ External influences (network or other connectivity problems).
 - ♦ Problems in the operating environment (low disk space or system errors)
 - ♦ Problems or changes in the data the e*Way is processing.
 - ♦ Corrections required to Collaboration Rule scripts that become evident in the course of normal operations.

One of the most important tools in the troubleshooter's arsenal is the e*Way log file. See the *e*Gate Integrator Alert and Log File Reference Guide* for an extensive explanation of log files, debugging options, and using the Schema Manager to monitor operations and performance.

Operational Overview

This chapter contains an overview of the operational behavior and basic internal processes of the PeopleSoft Batch e*Way, and covers the following topics:

e*Gate to PeopleSoft on page 51

PeopleSoft to e*Gate on page 54

Batch Operation on page 56

ODBC Operation on page 60

e*Way Architecture on page 63

Basic e*Way Processes on page 65

The information on Batch and ODBC operation is excerpted from the *Batch e*Way Intelligent Adapter User's Guide* and the *e*Way Intelligent Adapter for ODBC User's Guide*, respectively.

5.1 e*Gate to PeopleSoft

An e*Gate-outbound batch-mode interface loads a collection of transactions, usually in the form of a data file, into PeopleSoft. The interface can be triggered by a user or the source application, or by a scheduling mechanism (for example, a nightly synchronization of inventory information with an external system, or a weekly load of timecard information into PeopleSoft Payroll).

A data file from the source application is passed to e*Gate for transformation, and then loaded directly into the PeopleSoft application tables using SQL commands from the ODBC e*Way. The main components of this interface include:

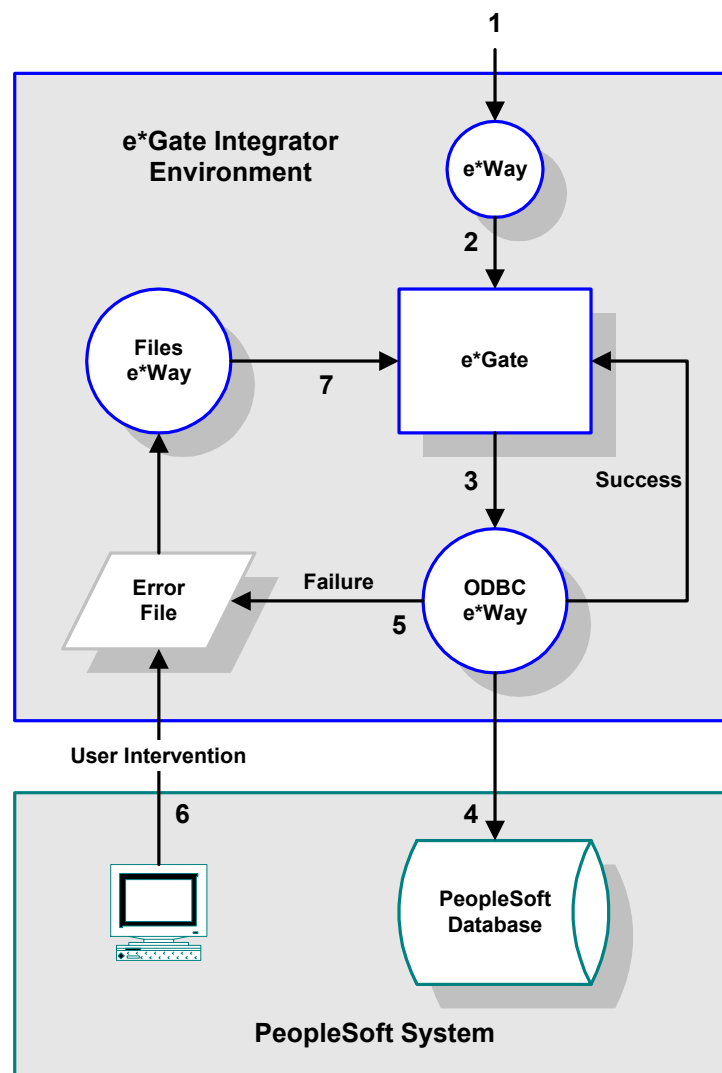
- A source-specific e*Way, to transfer a source data file.
- e*Gate Integrator, to route the data.
- An ODBC e*Way, to transform and load the data into the PeopleSoft system
- An error-file-handling e*Way, to process failed transactions.

5.1.1 Outbound Process Flow

The e*Gate-outbound process is as follows (see Figure 21):

- 1 A trigger Event or schedule in the source application initiates the creation of an interface file.
- 2 When the interface file is complete, the source e*Way sends the entire data file to *Gate.
- 3 For each row of data in the interface file, e*Gate routes the message to the ODBC e*Way.

Figure 21 e*Gate-to-PeopleSoft Batch Process Flow



- 4 Each message is transformed and mapped to the PeopleSoft-inbound message structure. The message structure contains one or more repeating table structures that are required to insert the transaction into the PeopleSoft application tables. For

example, an employee hire from the source system might require multiple row inserts into multiple PeopleSoft tables.

Once the message structure is loaded, the data is inserted into the application tables and a **commit** can be issued. The **commit** is configurable. After the message is processed successfully, the Batch e*Way is ready for the next message.

- 5 If the transformation fails or the data is not inserted, the original, un-transformed message from e*Gate is written to an error file.

Depending on the error handling requirements, when an error is encountered the interface may stop processing, stop processing and rollback all transactions, or bypass the error and continue processing.

- A If **stop on error** processing is desired, the Batch e*Way is suspended so that no more transactions are sent to PeopleSoft. Any unprocessed transactions from the interface file remain in the e*Gate queue.
 - B If **continue on error** processing is desired, the e*Way is not suspended and after the failed message is written to the error file, the remaining messages in the queue continue normal processing. Additional failed transactions are appended to the error file. When all of the transactions in the interface data file have been processed, the user can then correct the failed records for reprocessing.
 - C If **rollback on error** processing is desired, the Batch e*Way is suspended and the messages in the e*Gate queue are purged. Note that for rollback on error processing, only one **commit** is issued after all transactions in the interface file have been processed successfully.
- 6 At this point user intervention is required to edit the error file, using a standard file editor, and correct the offending data.
 - 7 Once the file is corrected, it is moved to the error handling directory where the error handling e*Way loads and reprocesses the messages. The corrected messages are routed to the Batch e*Way for transformation, mapping, and loading.
 - 8 For **stop on error** processing, once the failed message as been reprocessed successfully, the Batch e*Way is re-started to continue processing the remaining transactions in the e*Gate queue.

5.2 PeopleSoft to e*Gate

An e*Gate-inbound batch-mode interface extracts data from the PeopleSoft application tables, processes the data, and sends the results to an interface table or file. The data in the interface table or file is then sent to e*Gate for transformation and routing. Two different methods can be used in this mode:

- ODBC (Direct Load)
- Batch

The main components of this interface include:

- An e*Way (either ODBC or Batch), to retrieve a data table or file.
- e*Gate Integrator, to transform and route the data.
- A target-system-specific e*Way, to load the transformed data into the target system.

The design for outbound batch mode interfaces was created to be as flexible as possible. Most batch interfaces run on a scheduled basis, but the design also allows for an interface to be initiated by a remote function call (RFC) within the interface program. The design can accommodate interface data written to either an interface table or an interface file.

Outbound batch mode interfaces vary widely depending on the source and target applications, and the data to be interfaced. Because of this, the design does not attempt to generalize the extraction of data from PeopleSoft, the processing of that data, or output to a file or table. Instead, the design focuses solely on the process of sending the data from an interface table or file to the Intelligent Queue.

It begins after the interface program has loaded the data to the table or file, and ends once the data is transformed and routed to the Intelligent Queue. The design does not address a significant portion of the interfacing task, including logical units of work and target system error handling and alerting. These issues need to be addressed both as part of the interface program design and the inbound batch interface design for the target system.

5.2.1 Inbound Process Flow

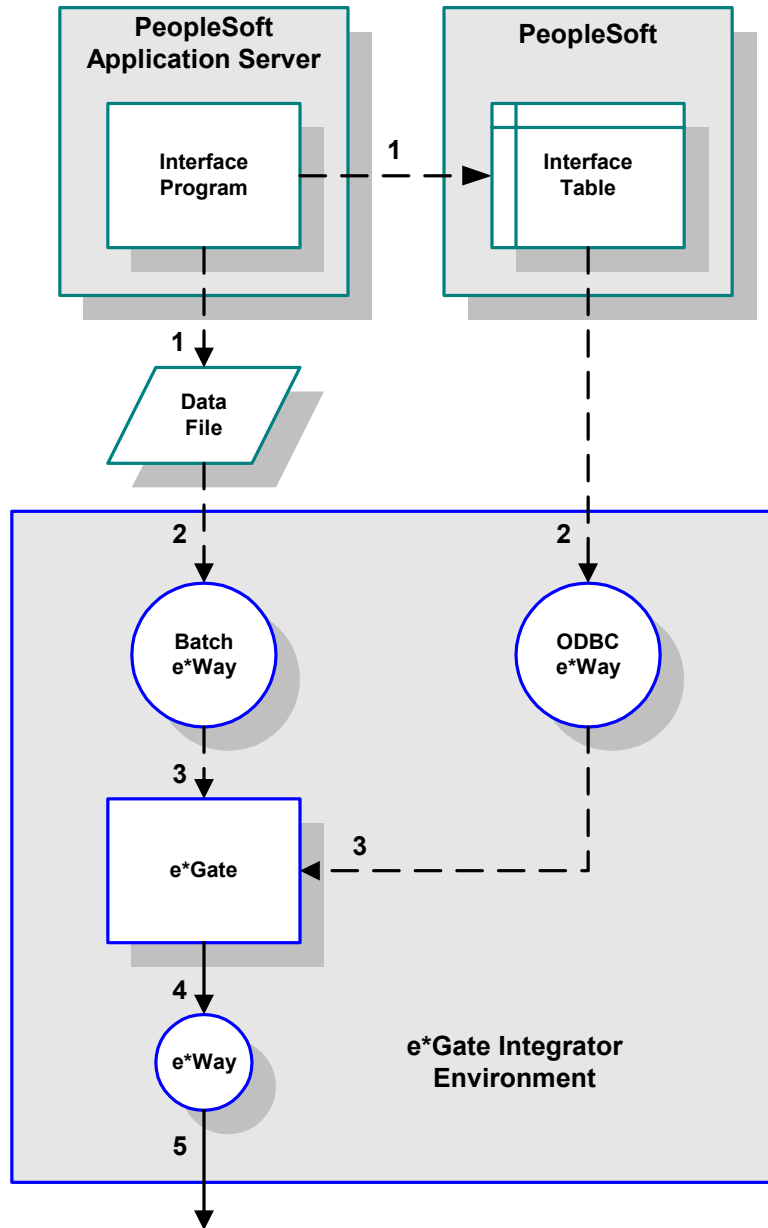
The e*Gate-inbound process is as follows (see Figure 22):

- 1 A process scheduler triggers the interface program to execute and writes data to an interface table or file (see Note 1).
- 2 The Batch e*Way retrieves the interface data.
- 3 The data is received by e*Gate.
- 4 Each row of data is identified, transformed, and routed to the target e*Way.
- 5 The transformed messages are loaded into the target application.

Note: *The interface program is a separate program outside of PeopleSoft that extracts data from the PeopleSoft application tables, manipulates the data, and writes the results to an output. The batch interface programs are usually initiated on the application*

server by a scheduling tool such as the PeopleSoft Process Scheduler, Autosys, or a 'cron job'.

Figure 22 PeopleSoft-to-e*Gate Process Flow



The Batch e*Way can be triggered to fetch data on a scheduled basis or by an external function call within the interface program. The polling schedule is a configurable parameter in the e*Gate workbench. The external function call is a function developed as part of the design that a developer can use within the interface program to send the data to the e*Gate.

5.3 Batch Operation

Note: For more complete information, see the *Batch e*Way Intelligent Adapter User's Guide*.

The PeopleSoft Batch e*Way makes use of the following Batch e*Way behavior models:

Subscriber

In this case, the e*Way polls an external system based on a schedule, and searches for files based on specific criteria. It then retrieves the files that match the criteria, stores them locally, and then reads the records in the files, while simultaneously keeps track of its own progress by maintaining state information in a separate file.

Message-Based

This model requires the use of the parameter, **Enable Message Configuration**. If this parameter is enabled, then the e*Way has a subscription to an ordering transmission that determines its activity. This subscription is to an XML message, with all relevant parameters governing the transfer, including the file to be sent (if it is an outbound transfer). There are two options for the ordering transmission.

- ◆ Receive one time (from one or more sources)
- ◆ Receive multiple times, according to a schedule that remains in effect until a new order arrives

5.3.1 Message-Based Operation

General Information

The *Order* Extensible Markup Language (XML) message has the following basic format:

```

<batch_e*Way_order>
  <command>                (command)    </command>
  <order_record>
    <error_record>
      </error_record>
    </order_record>
  <order_record>
    <error_record>
      </error_record>
    </order_record>
  <payload>                (DATA)      </payload>
</Batch_e*Way_Order>

```


The main record has the following subrecords:

- **Command**, which should be set to **Receive**
- **Order**, which contains the details for retrieving from a single source
- **Error**, which contains error information published by the e*Way after attempting to execute the order. This subrecord is only sent if the **Publish Status Record on Error parameter** is set to **Yes**.
- **Payload**, which is not used in **Receive** mode

The data can be the following forms:

- In the first case, the payload node can contain base64 data, in which case it has a **payload** attribute set to **base64Insitu**
- In the second case, the payload node represents the directory for the payload, in which case it has a **payload** attribute equal to **localDir**

Receiving Data with a *Receive* Order

Receiving from a file is similar to sending, as shown in this example.

```
<batch_e*way_order>
  <command>           receive           </command>
  <external_host_setup>
    <host_type>       Unix               </host_type>
    <user_name>       Alincoln          </user_name>
    <encrypted_password>liasdfLIJB       </encrypted_password>
    <file_transfer_method> ftp         </file_transfer_method>
    <return_tag>      Factor order      </return_tag>
  </external_host_setup>
  <communication_setup>
    <down_timeout>    10                 </down_timeout>
    <up_timeout>      20                 </up_timeout>
    <resend_timeout>  20                 </resend_timeout>
  </communication_setup>
  <subscribe_to_external>
    <remote_directory_name> /usr/home/honest_abe/from
    </remote_directory_name>
    <remote_file_regexp> Y*.dat        </remote_file_regexp>
  </subscribe_to_external>
</batch_e*way_order>
```

In this case, the e*Way retrieves all of the files in the designated directory that match the given regular expression, and stores them in a temporary directory. It then reads the entire contents of each file and sends it to e*Gate as a publication (using the **event-send-to-egate** function). The message sent is similar to the XML message that initiated the transfer, except for two differences:

- There is one return message per *order* in the command, instead of one return per *command*. Thus, if a command is received with orders for three transfers, the e*Way attempts three transfers, and returns the three retrieved files as three *receive* responses.
- It contains a *payload* field that contains the data received. Please refer to the following example:

```

<batch_e*way_order>
  <command>          receive          </command>
  <external_host_setup>
    <host_type>      Unix          </host_type>
    <user_name>      Alincoln       </user_name>
    <encrypted_password> liasdfLIJB </encrypted_password>
    <file_transfer_method> ftp      </file_transfer_method>
    <return_tag>     Factor order </return_tag>
  </external_host_setup>
  <communication_setup>
    <down_timeout>   10           </down_timeout>
    <up_timeout>     20           </up_timeout>
    <resend_timeout> 20           </resend_timeout>
  </communication_setup>
  <subscribe_to_external>
    <remote_directory_name> /usr/home/honest_abe/from
    </remote_directory_name>
    <remote_file_regexp> Y*.dat      </remote_file_regexp>
  </subscribe_to_external>
  <payload>          (DATA)        </payload>
</batch_e*way_order>

```

Only when it has sent all records does the e*Way acknowledge (ACK) the *order* command message. The **<return_tag>** field of the XML message is used to store a unique tag generated by the originator of the command. This tag helps the e*Gate system administrator to determine, as each response comes back, which system gave that response.

As a final example of the receive command, consider this example of a command to go to three different systems for three different kinds of data, Factory Orders, Builds of Materials, and Engineering Updates. First, note the command record (transfer details omitted for brevity):

```

<batch_e*way_order>
  <command>          receive          </command>
  <return_tag>      Factor order       </return_tag>
  <return_tag>      Build of Materials </return_tag>
  <return_tag>      Engineering Updates </return_tag>
</batch_e*way_order>

```

The Batch e*Way attempts each *receive* transfer, and follows its normal procedures for retrying and raising exceptions if problems occur. As each transfer succeeds, it returns an XML message with the *payload* and the corresponding return tag. If it fails, it returns an XML message with the error record.

```

<batch_e*way_order>
  <command>          receive          </command>
  <return_tag>      Factor order       </return_tag>
  <payload>         (DATA)            </payload>
</batch_e*way_order>

```

And then each of the other two, (Build of Materials, and Engineering Updates).

```

<batch_e*way_order>
  <command>          receive          </command>
  <return_tag>      Build of Materials </return_tag>
  <payload>         (DATA)            </payload>
</batch_e*way_order>

```

```

<batch_e*way_order>
  <command>           receive           </command>
    <return_tag>      Engineering Updates </return_tag>
    <payload>         (DATA)           </payload>
</batch_e*way_order>

```

Please see [Receive XML Messages](#) on page 120 for the corresponding Document Type Definition (DTD) file.

Error Reporting

If the parameter **Publish Status Record on Error** is set to **Yes**, and the e*Way encounters problems with one order, it publishes the command message containing only the orders that failed, along with the corresponding error records. This is illustrated in the following template:

```

<batch_e*way_order>
  <command>           (command)           </command>
    <order_record>
      <error_record>
        <error_code>           </error_code>
        <error_text>          </error_text>
        <last_action>         </last_action>
      </error_record>
    </order_record>
    <payload>         (DATA)           </payload>
</batch_e*way_order>

```

The *last action* record contains whatever command the e*Way can indicate. Thus, if a failure occurs on renaming a file after the transfer, then the e*Way populates this field with the *rename* command it is attempting to perform.

Please see [Error Messages](#) on page 121, for the corresponding Document Type Definition (DTD) file.

5.4 ODBC Operation

Note: For more complete information, see the *e*Way Intelligent Adapter for ODBC User's Guide*.

5.4.1 Using SQL Functions

Static vs. Dynamic SQL Functions

Dynamic SQL statements are built and executed at run time versus Static SQL statements that are embedded within the program source code. Dynamic statements do not require knowledge of the complete structure on an SQL statement before building the application. This allows for run time input to provide information about the database objects to query.

The application can be written so that it prompts the user or scans a file for information that is not available at compilation time.

In Dynamic statements the four steps of processing an SQL statement take place at run time, but they are performed only once. Execution of the plan takes place only when EXECUTE is called. **Figure 23 on page 61** shows the difference between Dynamic SQL with immediate execution and Dynamic SQL with prepared execution. See Static SQL Functions and Dynamic SQL Functions in the *e*Way Intelligent Adapter for ODBC User's Guide*.

Benefits of Dynamic SQL

Using dynamic SQL commands, an application can prepare a *generic* SQL statement once and execute it multiple times. Statements can also contain markers for parameter values to be supplied at execution time, so that the statement can be executed with varying inputs.

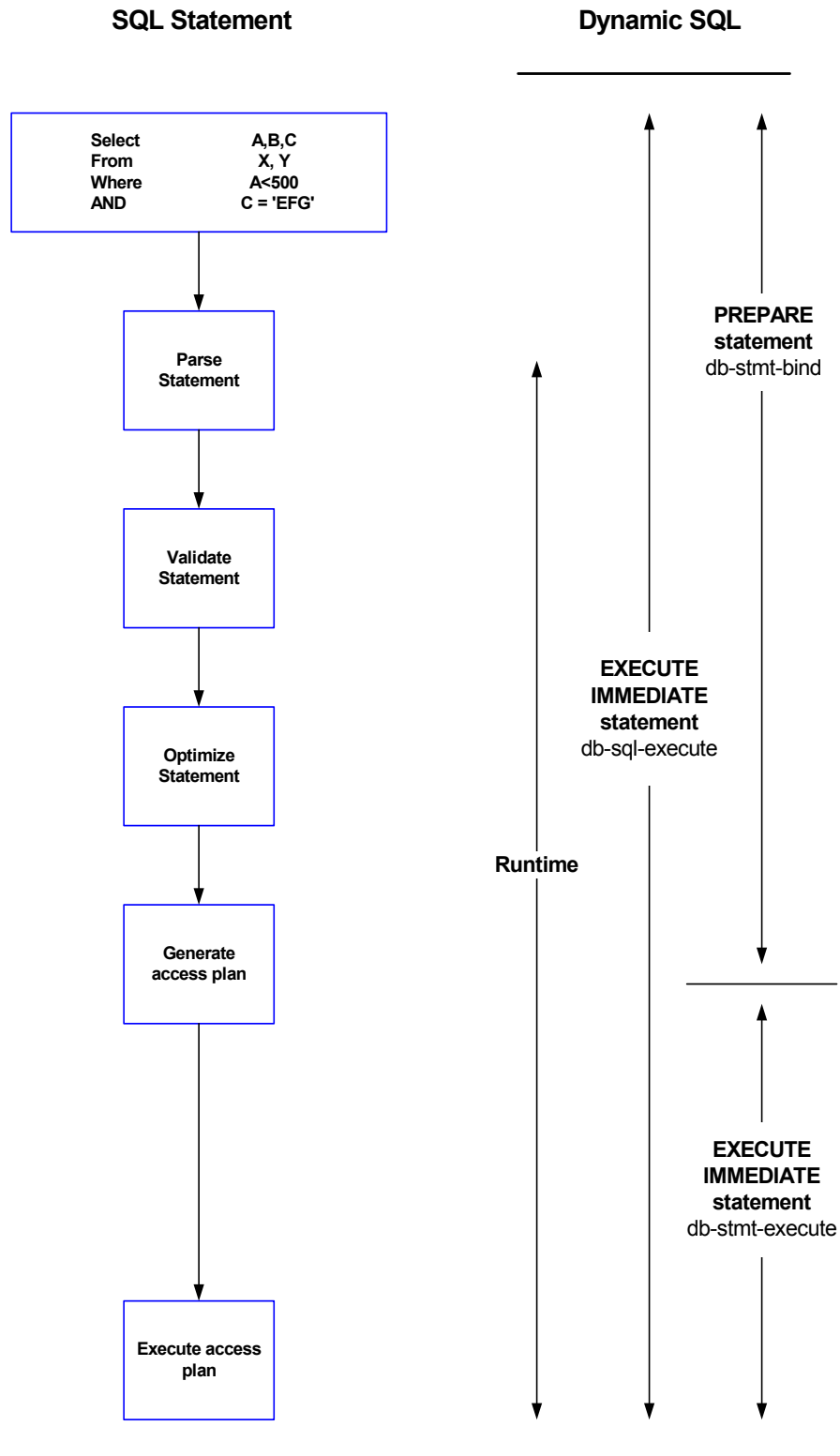
Limitations of Dynamic SQL

The use of dynamic SQL commands has some significant limitations. A dynamic SQL implementation of an application generally performs worse than an implementation where permanent stored procedures are created and the client program invokes them with RPC (remote procedure call) commands.

Benefits of Stored Procedures

When a stored procedure is created for an application, SQL statement compilation and optimization are performed once when the procedure is created. With a dynamic SQL application, compilation and optimization are performed every time the client program runs. A dynamic SQL implementation also incurs database space overhead because each instance of the client program must create separate compiled versions of the application's prepared statements. When you design an application to use stored procedures and RPC commands, all instances of the client program can share the same stored procedure. See Stored Procedure Functions in the *e*Way Intelligent Adapter for ODBC User's Guide*.

Figure 23 Example of Dynamic SQL processing



ODBC SQL Type Support

The following table shows the supported SQL data types and the corresponding native data type for the database.

Table 11 ODBC SQL Type Support

SQL Type Name	SQL Datatype	Oracle Datatype
SQL_BIT	BIT	N/A
SQL_BINARY	BINARY (n)	N/A
SQL_VARBINARY	VARBINARY (n)	RAW (n)
SQL_CHAR	CHAR (n)	CHAR (n)
SQL_VARCHAR	VARCHAR (n)	VARCHAR2 (n)
SQL_DECIMAL	DECIMAL (p, s)	NUMBER (p, s)
SQL_NUMERIC	NUMERIC (p, s)	N/A
SQL_TINYINT	TINYINT	+
SQL_BIGINT	BIGINT	+
SQL_SMALLINT	SMALLINT	+
SQL_INTEGER	INTEGER	+
SQL_REAL	REAL	*
SQL_FLOAT	FLOAT(p)	FLOAT(b)
SQL_DOUBLE	DOUBLE PRECISION	FLOAT
SQL_DATE	DATE	N/A
SQL_TIME	TIME	N/A
SQL_TIMESTAMP	TIMESTAMP	DATE
SQL_LONGVARCHAR	LONG VARCHAR	LONG
SQL_LONGVARBINARY	LONG VARBINARY	LONG RAW

*Oracle float (p) specifies a floating point number with precision range from 1 to 126.

+Oracle uses number (p) to define data types that span TINYINT, BIGINT, SMALLINT, and INTEGER. Oracle **int** type is internally mapped to NUMBER (38) which is returned as SQL_DECIMAL.

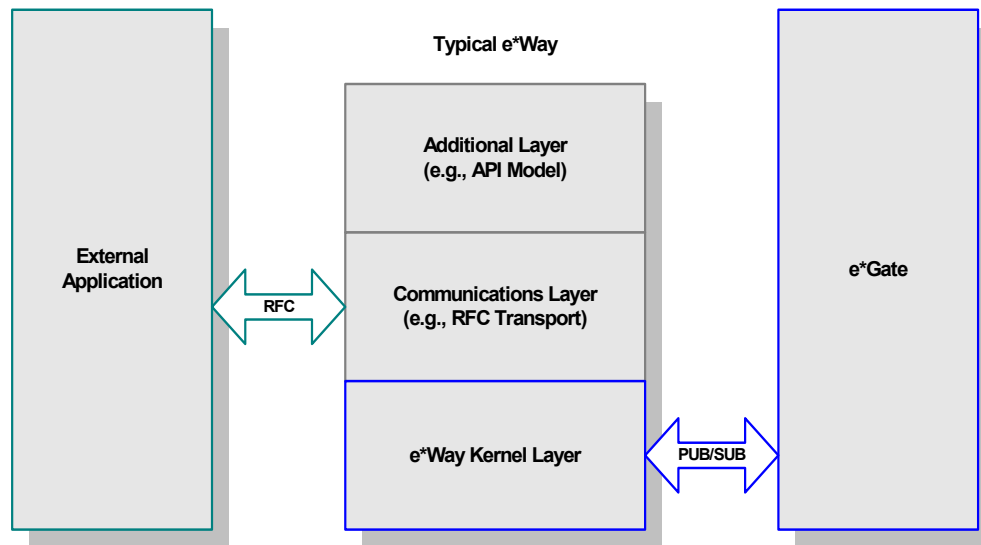
Note: All variable precision data types require precision values.

SQL_DECIMAL and SQL_NUMERIC data types require specification of scale which indicates the number of digits to the right of the decimal point.

5.5 e*Way Architecture

Conceptually, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers (see Figure 24). 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).

Figure 24 Typical e*Way Architecture



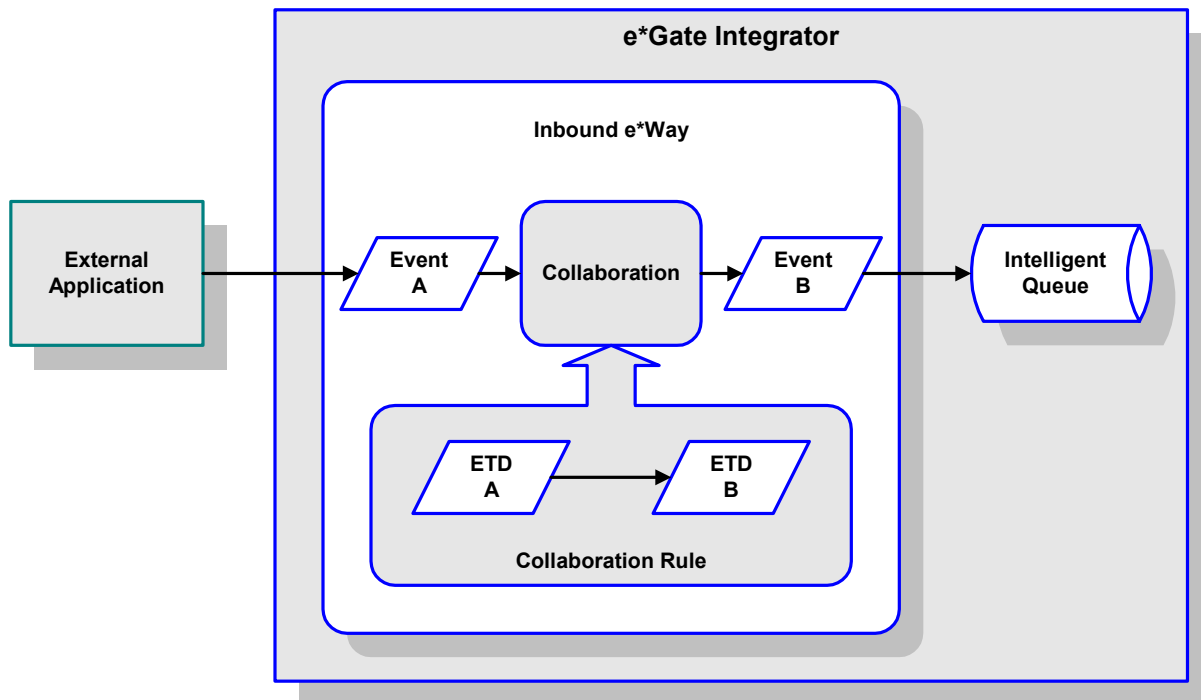
The upper layers of the e*Way use Monk functions to perform Business Process modeling and ETD mapping, package data as e*Gate *Events*, send those Events to Collaborations, and manage interaction with the external system. These layers are built upon an e*Way Kernel layer that manages the basic operations of the e*Way, data processing, and communication with other e*Gate components.

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

5.6 Event Type Definitions and Collaborations

Collaborations execute the business logic that enable the e*Way to do its intended work. In turn, each Collaboration executes a Collaboration Rule, containing the actual instructions to execute the business logic. Each Collaboration that publishes its processed Events internally (within e*Gate Integrator) requires one or more IQs to receive the Events, as shown in Figure 25. Any Collaboration that publishes its processed Events only to an external system does *not* require *any* IQs.

Figure 25 Collaborations and IQs



Configuration options that control the Monk environment and define the Monk functions used to perform various e*Way operations are discussed in [Chapter 6](#). You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft Word* or *Notepad*, or UNIX *vi*). The available set of e*Way API functions is described in [Chapter 7](#). 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.

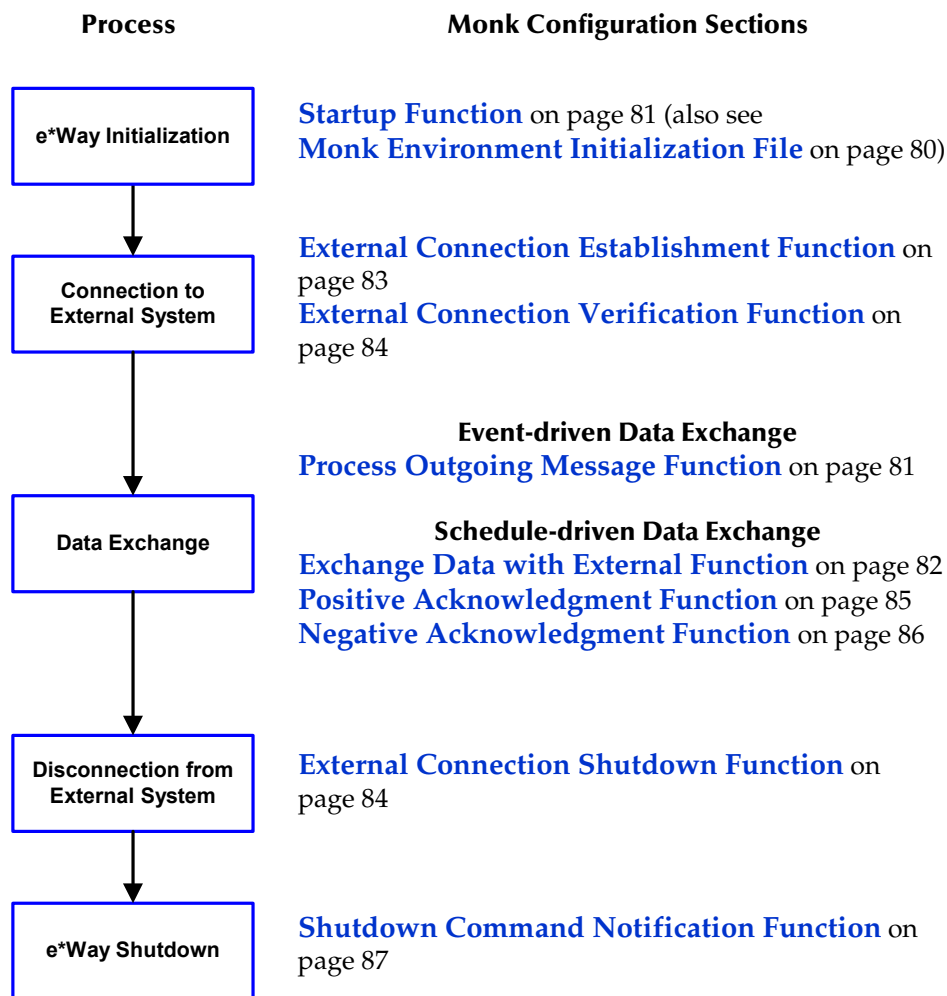
For more information on defining Collaborations, defining IQs, assigning Collaborations to e*Ways, or configuring Collaborations to publish Events, see the *e*Gate Integrator User's Guide*.

5.7 Basic e*Way Processes

Note: This section describes the basic operation of a typical e*Way based on the Generic e*Way Kernel. Not all functionality described in this section is used routinely by this e*Way.

The most basic processes carried out by an e*Way are listed in Figure 26. In e*Ways based on the Generic Monk e*Way Kernel (using `stcewgenericmonk.exe`), these processes are controlled by the listed Monk functions. Configuration of these functions is described in the referenced sections of this User's Guide.

Figure 26 Basic e*Way Processes

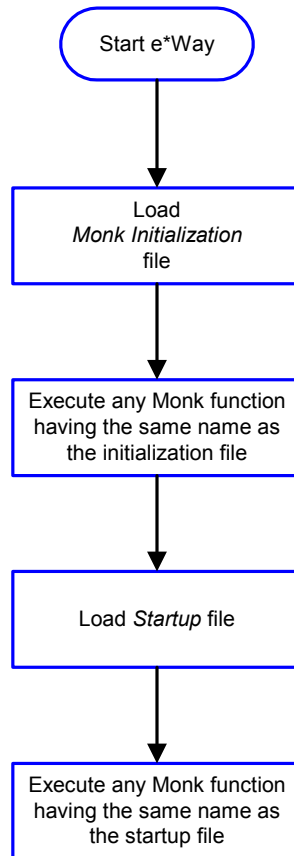


A series of diagrams on the next several pages illustrate the interaction and operation of these functions during the specified processes. Configuring the parameters associated with these functions is covered in [Chapter 6](#), while the functions themselves are described in [Chapter 7](#).

Initialization Process

Figure 27 illustrates the e*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

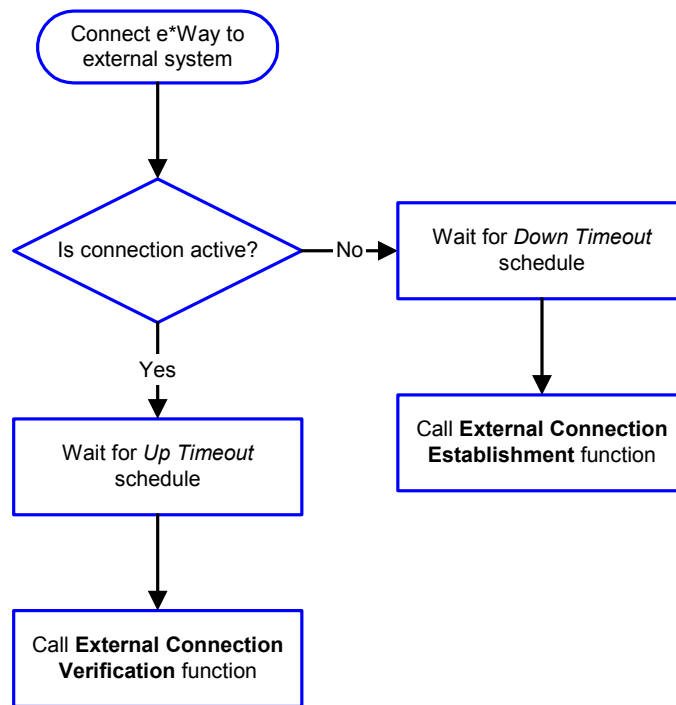
Figure 27 Initialization Process



Connect to External Process

Figure 28 illustrates how the e*Way connects to the external system, using the **External Connection Establishment Function** and **External Connection Verification Function**.

Figure 28 Connection Process



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 30 on page 69** and **Figure 29 on page 68** 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 117 and **send-external-down** on page 117 for more information.

Data Exchange Process

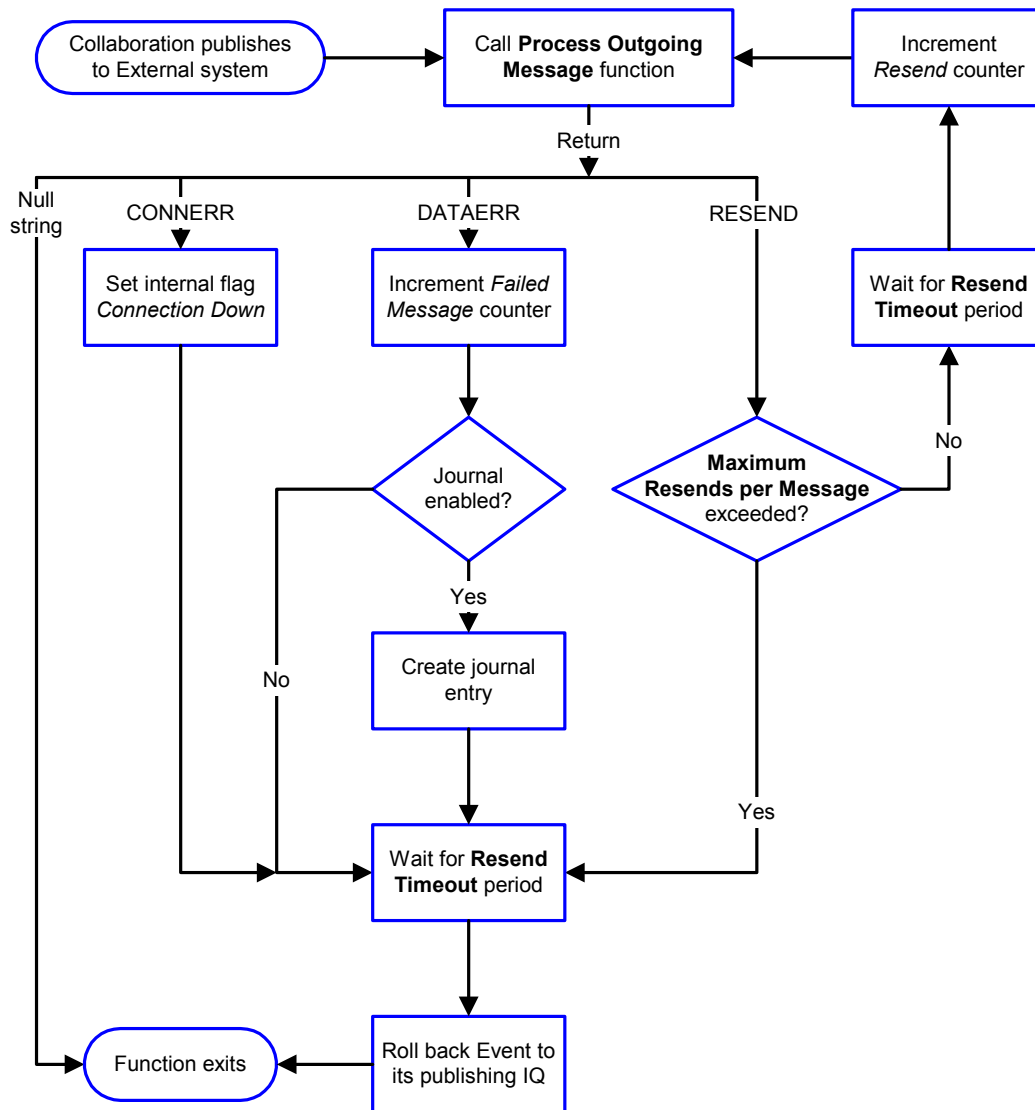
Event-driven

Figure 29 illustrates how the e*Way's event-driven data exchange process works, using the **Process Outgoing Message Function**.

The e*Way periodically 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.

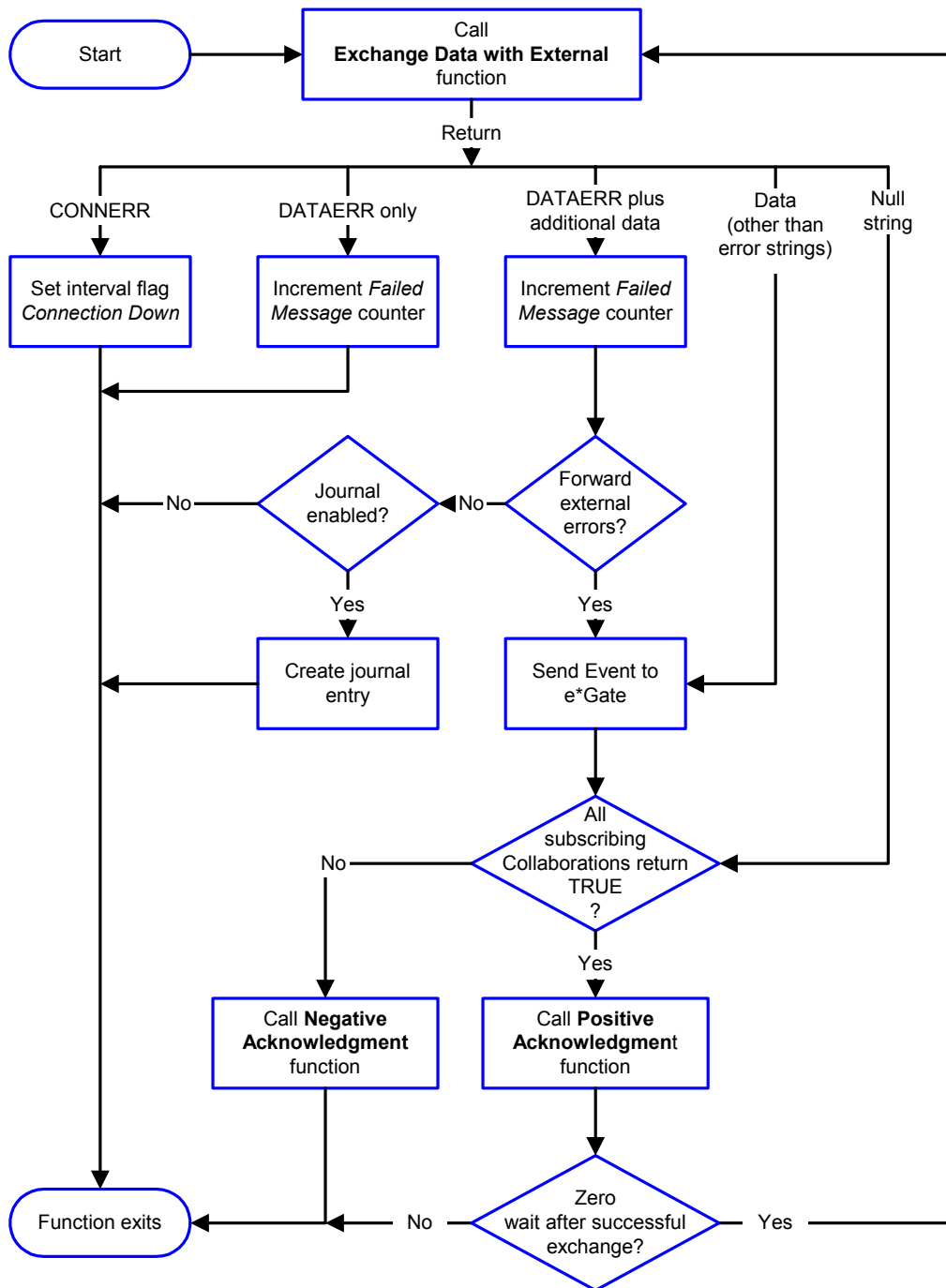
Figure 29 Event-Driven Data Exchange Process



Schedule-driven

Figure 30 illustrates how the e*Way's schedule-driven data exchange process works for incoming data, using the **Exchange Data with External Function, Positive Acknowledgment Function**, and **Negative Acknowledgment Function**.

Figure 30 Schedule-Driven Data Exchange Process



Start can occur in any of the following ways:

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

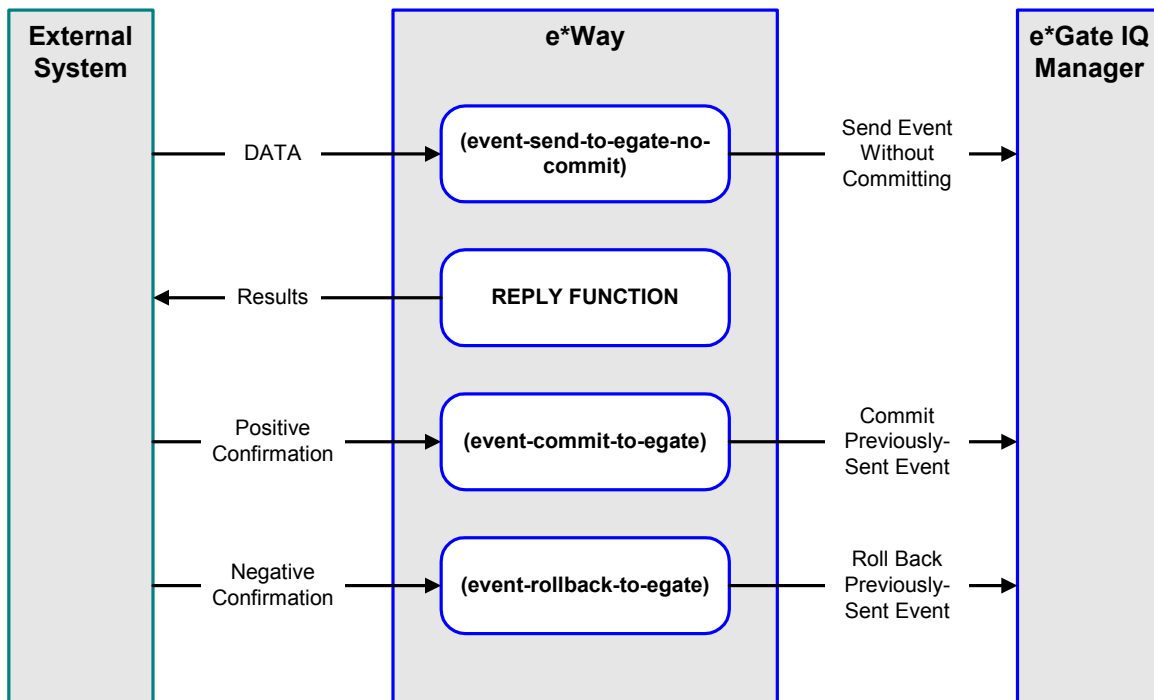
*Send Events to e*Gate* can be implemented using any of the following Monk functions:

- **event-send-to-egate**
- **event-send-to-egate-ignore-shutdown**
- **event-send-to-egate-no-commit**

The last of these is used when confirmation of correct transmission is required from the external system. In this case, the e*Way sends information back to the external system after receiving data. Depending upon whether the acknowledgment is positive or negative, you subsequently use one of the following functions to complete the process (see Figure 31):

- **event-commit-to-egate**
- **event-rollback-to-egate**

Figure 31 Send Event to e*Gate with Confirmation

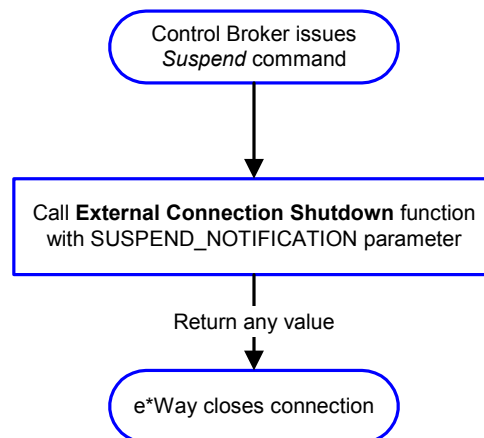


After the function exits, the e*Way waits for the next *Start* time or command.

Disconnect from External Process

Figure 32 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

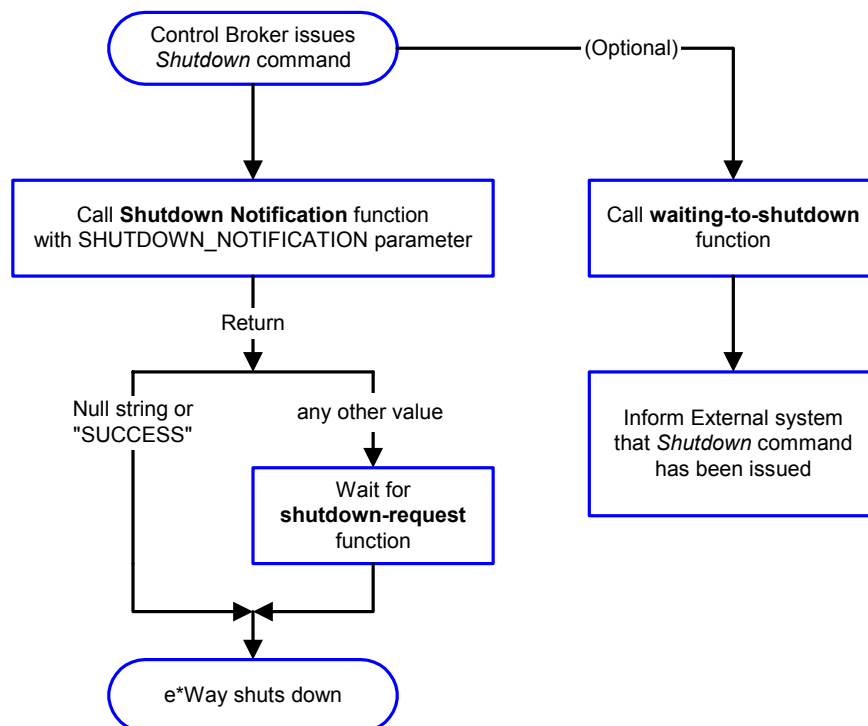
Figure 32 Disconnect Process



Shutdown Process

Figure 33 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

Figure 33 Shutdown Process



Configuration Parameters

This chapter describes the configuration parameters for the e*Way Intelligent Adapter for PeopleSoft Batch.

6.1 Overview

The e*Way's configuration parameters are set using the e*Way Editor; see [Configuring the e*Way](#) on page 41 for procedural information. The default configurations are provided in `batch.def` and `dart.def`.

The following configuration sections are included in this User's Guide, since they are an integral part of the operational description. For all other configuration information, please refer to the *Batch e*Way Intelligent Adapter User's Guide* or the *e*Way Intelligent Adapter for ODBC User's Guide*, as applicable.

- [General Settings](#) on page 73
- [Communication Setup](#) on page 75
- [Monk Configuration](#) on page 79
- [Dynamic Configuration](#) on page 88

6.2 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 Journalled 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
-

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. When this maximum is reached, the e*Way waits for the number of seconds specified by the [Resend Timeout](#) parameter, and then rolls back the Event to its publishing IQ.

Required Values

An integer from 1 through 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 shuts down and exits.

Required Values

An integer from 1 through 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 are queued to the e*Way’s configured queue. See [Exchange Data with External Function](#) on page 82 for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages are not to be forwarded. See [Data Exchange Process](#) on page 68 for more information about how the e*Way uses this function.

6.3 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 e*Gate Schema Designer controls when the e*Way executable runs. The schedule that 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.*

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 from 0 through 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 setting of this parameter is ignored and the e*Way invokes the [Exchange Data with External Function](#) immediately
- If it is desired to invoke the [Exchange Data with External Function](#) again as soon as possible when data is **not** queued to e*Gate via the return mechanism, the e*Way Kernel Monk function [insert-exchange-data-event](#) can be called directly (prior to leaving the exchange function) to accomplish this
- If this parameter is set to zero, no exchange data schedule is set and the [Exchange Data with External Function](#) is never called

See also

[Start Exchange Data Schedule](#) on page 76

[Stop Exchange Data Schedule](#) on page 77

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, the e*Way invokes the **Exchange Data with External Function** immediately
- If it is desired to invoke the **Exchange Data with External Function** again as soon as possible when data is **not** queued to e*Gate via the return mechanism, the e*Way Kernel Monk function **insert-exchange-data-event** can be called directly (prior to leaving the exchange function) to accomplish this
- 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**

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, regular, interval (such as weekly, daily, or every *n* seconds)

Other Requirements

If you set a schedule using this parameter, you must also define *all* of the following parameters. If you do not, the e*Way terminates execution when the schedule attempts to start.

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

Additional Information

When the schedule starts, the e*Way determines whether or not:

- it is waiting to send an ACK or NAK to the external system (using the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**)
- 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.

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, regular, interval (such as weekly, daily, or every n seconds)

Down Timeout

Description

Specifies the number of seconds for the e*Way to wait between calls to the **External Connection Establishment Function**.

Required Values

An integer from 1 through 86,400. The default is 15.

Up Timeout

Description

Specifies the number of seconds for the e*Way to wait between calls to the **External Connection Verification Function** to verify that the connection is still up.

Required Values

An integer from 1 through 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 from 1 through 86,400. The default is 15.

Exchange-if-in-window-on-startup

Description

If this Batch e*Way parameter is set to Yes and e*Way starts within an *exchange data* window (see [Exchange Data Interval](#)), the e*Way will invoke the [Exchange Data with External Function](#) parameter immediately.

Required Values

Yes or No. The default is No.

6.4 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.

Specifying Function or File Names

Parameters that require the name of a Monk function accept either a function name (implied by the absence of a period <.>) or the name of a file (optionally including path information) containing a Monk function. If a file name is specified, the function invoked is given by the base name of the file (for example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**). If path information is specified, that path is appended to the **Load Path**.

If you specify a file name, be sure that the file has one of the following extensions:

- .monk
- .tsc
- .dsc

Specifying Multiple Directories

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.

Load Path

The Monk *load path* is the path Monk uses to locate files and data (set internally within Monk). The default load paths are determined by the **SharedExe** and **SystemData** settings in the **.egate.store** file. See the *e*Gate Integrator System Administration and Operations Guide* for more information about this file.

Additional Path

Description

Specifies a path to be appended to the **Load Path**. A directory specified here is searched *after* searching the default load path.

Required Values

A pathname, or a series of paths separated by semicolons.

Note: For *direct-load* or *direct-extract* operation, this parameter is optional and may be left blank; for *batch-mode* operation, this parameter is not used.

Additional information

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 loaded automatically into the e*Way's Monk environment.

Required Values

A pathname, or a series of paths separated by semicolons.

- For ODBC (direct-load/direct-extract) operation, the default value is **monk_library/dart**.
- For batch-mode operation, the default value is **monk_library/batch**.

Note: This parameter is optional and may be left blank.

Monk Environment Initialization File

Description

Specifies a file that contains environment initialization functions, which is loaded after the [Auxiliary Library Directories](#) are loaded.

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.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-init**.
- For batch-mode operation, the default value is **batch-init**.

Note: This parameter is optional and may be left blank.

Returns

The string "FAILURE" indicates that the function failed, and the e*Way exits; any other string, including a *null string*, indicates success.

Additional information

- 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); it is 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

- The e*Way loads this file and try to invoke a function of the same base name as the file name

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. It is called after the e*Way loads the specified **Monk Environment Initialization File** and any files within the specified **Auxiliary Library Directories**. This function accepts no input, and must return a string.

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 containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-startup**.
- For batch-mode operation, the default value is **batch-startup**.

Note: This parameter is optional and may be left blank.

Returns

The string "FAILURE" indicates that the function failed, and the e*Way exits; any other string (including a *null string*) indicates success.

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, rather than schedule-driven). The function requires a non-null string as input (i.e., the outgoing Event to be sent), and must return a string.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-proc-outgoing**.
- For batch-mode operation, the default value is **batch-proc-out**.

Note: This parameter is **required**, and must **not** be left blank.

Returns

- A *null string* ("") indicates that the Event was published successfully to the external system

- A string beginning with **RESEND** indicates that the Event should be resent
- A string beginning with **CONNERR** indicates that there is a problem with the connection to the external system, and causes a rollback of the Event
- A string beginning with **DATAERR** indicates that there is a problem with the message (Event) data itself, and causes a rollback of the Event
- A string beginning with **SHUTDOWN** indicates that the e*Way must exit immediately
- If any string other than one of the preceding is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function

Additional Information

- The e*Way invokes this function when one of its Collaborations publishes an Event to an *external* destination (as specified within the e*Gate Schema Designer).
- Once this function has been called with a *non-null string*, the e*Way does not process another Event until the current Event has been completely processed.

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

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is invoked automatically by the **Start Exchange Data Schedule** or manually by the **start-schedule** Monk function, and is responsible for either sending data to or receiving data from the external system. If this function returns data, it is queued to e*Gate in an inbound Collaboration. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-data-exchg**.
- For batch-mode operation, the default value is **batch-exchange-data**.

Note: *This parameter is **conditional** and must be supplied only if the **Exchange Data Interval** is set to a non-zero value.*

Returns

- A *null string* ("") indicates that the data exchange was completed successfully, but with no resultant data sent back to the e*Gate system

- A string beginning with **CONNERR** indicates that there is a problem with the connection to the external system
- A string beginning with **DATAERR** indicates that there is a problem with the message (Event) data itself. If the error string contains data beyond the keyword, the entire string is queued to e*Gate if an inbound Collaboration is so configured and **Forward External Errors** is set to **Yes**. Queuing, however, is performed without the subsequent sending of a **ACK** or **NAK** to the external system.
- Any other string indicates that the contents of the string are packaged as an inbound Event

Additional Information

- Data can be queued directly to e*Gate by using the **event-send-to-egate** Monk function or, if a two-phase approach is required, by using **event-send-to-egate-no-commit** and then **event-commit-to-egate** or **event-rollback-to-egate** to commit or rollback the enqueued events, as appropriate

Note: *Until an Event is committed, it is not revealed to subscribers of that Event.*

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way calls (repeatedly) when it has determined that the connection to the external system is down. The function accepts no input and must return a string.

This function is executed according to the interval specified within the **Down Timeout** parameter, and is called *only* according to this schedule. Once the e*Way has determined that its connection to the external system is up, it calls the **External Connection Verification Function** (see next).

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-conn-estab**.
- For batch-mode operation, the default value is **batch-ext-connect**.

Note: *This parameter is **required**, and must **not** be left blank.*

Returns

- A string beginning with **SUCCESS** or **UP** indicates that the connection was established successfully
- A string beginning with **DOWN** indicates that the connection was not established successfully
- Any other string, including a *null string*, indicates that the attempt to establish the connection failed and the external state is unknown

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. It is executed according to the interval specified within the **Up Timeout** parameter, and is called *only* according to this schedule. The function accepts no input and must return a string.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

Note: This parameter is optional and may be left blank.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-conn-ver**.
- For batch-mode operation, the default value is **batch-ext-verify**.

Returns

- "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

Additional Information

If this function is not specified, the e*Way executes the **External Connection Establishment Function** in its place. This latter function also 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 calls to shut down the connection to the external system. This function is invoked only when the e*Way receives a *suspend* command from a Control Broker.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-shutdown**.
- For batch-mode operation, the default value is **db-stdver-conn-shutdown**.

Note: This parameter is **required**, and must **not** be left blank.

Input

A string indicating the purpose for shutting down the connection.

- "SUSPEND_NOTIFICATION" - the e*Way is being suspended or shut down

- “RELOAD_NOTIFICATION” - the e*Way is being reconfigured

Returns

A string, the value of which is ignored. Any return value indicates that the *suspend* command can proceed and that the connection to the external system can be broken immediately.

Note: *Include in this function any required “clean up” operations that must be performed as part of the shutdown procedure, but before the e*Way exits.*

Positive Acknowledgment Function

Description

This function is loaded during the initialization process and is called when all data received from the external system has been processed and enqueued successfully. The function requires a non-null string as input (the Event to be sent to e*Gate) and must return a string.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-pos-ack**.
- For batch-mode operation, the default value is **batch-ack**.

Note: *This parameter is **conditional** and must be supplied only if the **Exchange Data with External Function** is set to a non-zero value.*

Input

A string, the inbound Event to e*Gate.

Returns

- The string beginning with **CONNERR** indicates a problem with the connection to the external system; when the connection is re-established, the function is called again, with the same input data
- Any other string, including a *null string*, indicates that the acknowledgement has been sent to the external system successfully

Additional Information

- 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. The e*Way executes this function only if the Event’s processing is completed successfully by *all* the Collaborations to which it was sent; otherwise, the e*Way executes the **Negative Acknowledgment Function**.
- This function can return data to be queued, but the e*Way will *not* acknowledge the data with an **ACK** or **NAK**.

Note: *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

Negative Acknowledgment Function

Description

This function is loaded during the initialization process and is called when the e*Way fails to process or enqueue data received from the external system successfully. The function requires a non-null string as input (the Event to be sent to e*Gate) and must return a string.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-neg-ack**.
- For batch-mode operation, the default value is **batch-nak**.

Note: *This parameter is **conditional** and must be supplied only if the **Exchange Data with External Function** is set to a non-zero value.*

Input

A string, the inbound Event to e*Gate.

Returns

- The string beginning with **CONNERR** indicates a problem with the connection to the external system; when the connection is re-established, the function is called again, using the same input data
- Any other string, including a *null string*, indicates that the acknowledgement has been sent to the external system successfully

Additional Information

- This function is called only 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. The e*Way executes this function if the Event's processing is not completed successfully by *all* the Collaborations to which it was sent; otherwise, the e*Way executes the **Positive Acknowledgment Function**.
- This function can return data to be queued, but the e*Way will *not* acknowledge the data with an **ACK** or **NAK**.

Note: *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

Shutdown Command Notification Function

Description

The e*Way calls this Monk function automatically to notify the external system that it is about to shut down. This function also can be used to shut down the connection with the external. The function accepts a string as input and must return a string.

Required Values

The name of a Monk function or the name of a file containing a Monk function.

- For ODBC (direct-load/direct-extract) operation, the default value is **db-stdver-shutdown**.
- For batch-mode operation, the default value is **batch-shutdown-notify**.

Note: This parameter is **required**, and must **not** be left blank.

Input

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.

Returns

- 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

Additional Information

If you postpone a shutdown using this function, be sure to use the **shutdown-request** function to complete the process in a timely manner.

6.5 Dynamic Configuration

This section describes parameters used to configure a dynamic Batch e*Way:

Note: Use the following XML Document Type Definition (DTD) and e*Gate Event Type Definition (ETD) files when using dynamic configuration:

Table 12 Dynamic Configuration Files

DTD File	Corresponding ETD File
batch_eway_data.dtd	batch_eway_data.xsc
batch_eway_error.dtd	batch_eway_error.xsc
batch_eway_order.dtd	batch_eway_order.xsc

Enable Message Configuration

Description

Use this parameter to indicate that the e*Way contains an XML message which determines its activities. The XML message should contain all relevant parameters that govern the transfer. See [Appendix A](#) for details about the DTD.

Note: When the XML message sets the e*Way to receive, Batch retrieves the external file and wraps it into the XML payload (see [Data Message](#) on page 122), and transforms the data into Base64 format. To send the data back in its original format, use the **Base64-to-Raw** Monk function. Details on how to use this function are explained in the *Monk Developer's Reference*.

Required Values

Yes or No. (No is the default).

When this parameter is set to **Yes**, the Batch e*Way becomes Event-driven, so it does *not* exchange data based on scheduling, and the record type is always a single record. Additionally, certain other configuration parameters are affected, as shown in Table 13.

Note: If the fields marked as **Overridden by message** are set by the XML message, then the table below holds true. However, if the fields are *not* set by the XML message, then those fields marked as **Overridden by message** *must* be specified in the .cfg file. Only **subscriptions** fields must be set.

Table 13 Effect of Enabling Message Configuration

Section	Parameter	Effect
Communication Setup	Start Exchange Data Schedule	Ignored.
	Stop Exchange Data Schedule	Ignored.
	Exchange Data Interval	Ignored.
	Zero Wait Between Successful Exchanges	Ignored.
	Down Timeout	Ignored.
	Up Timeout	Ignored.
External Host Setup	External Host Name	Overridden by message.
	Host Type	Overridden by message.
	User Name	Overridden by message.
	Encrypted Password	Overridden by message.
	File Transfer Method	Overridden by message.
Monk Configuration	Process Outgoing Message Function	The XML event is parsed and processed.
	Exchange Data With External Function	Ignored.
	Positive Acknowledgment Function	Ignored.
	Negative Acknowledgment Function	Ignored.
	Startup Function	Normal behavior, but the value assigned to transfer method is ignored.
Publish To External	Remote Directory Name	Overridden by message.
Publish To External	Remote File Name	Overridden by message.
	Append or Overwrite when Transferring Files	Overridden by message.
	Record Type	Automatically set to Single Record . Any other value is ignored.
	Record Delimiter	Ignored.
	Delimiter on Last Record	Ignored.
	Record Size	Ignored.
	Remote Command After Transfer	Overridden by message.
	Remote Rename or Archive Name	Overridden by message.
	Local Command After Transfer	Overridden by message.
Local Archive Directory	Overridden by message.	

Table 13 Effect of Enabling Message Configuration (Continued)

Section	Parameter	Effect
Recourse Action	Action on Fetch Failure	Normal behavior, but an additional option is needed to publish the Event that contains the configuration message.
	Action on Send Failure	Normal behavior, but an additional option is needed to publish the Event that contains the configuration message.
Sequence Numbering	Starting Sequence Number	Ignored.
	Max Sequence Number	Ignored.
SOCKS	Server Host Name	Overridden by message.
	Server Port	Overridden by message.
Subscribe To External	Remote Directory Name	Overridden by message.
	Remote File Regexp	Overridden by message.
	Record Type	Ignored.
	Record Delimiter	Ignored.
	Delimiter on Last Record	Ignored.
	Record Size	Ignored.
	Remote Command After Transfer	Overridden by message.
	Remote Rename or Archive Name	Overridden by message.
	Local Command After Transfer	Overridden by message.
Local Archive Directory	Overridden by message.	
FTP	server_port	Overridden by message.
	mode	Overridden by message.
	Pretransfer_Commands	Overridden by message.
	Posttransfer_Commands	Overridden by message.

Publish Status Record on Success

Description

When this parameter is set to **Yes**, the Batch e*Way will publish a *good error* record to e*Gate, with the same format that is specified in `batch_eway_error.dtd`. (See [Error Messages](#) on page 121.) The *good error* record is published only when the payload has been successfully sent to the remote host.

Note: *This parameter is not used by the PeopleSoft Batch e*Way.*

Required Values

Yes or No. (No is the default).

Publish Status Record on Error

Description

This parameter determines whether or not the Batch e*Way publishes an error record to e*Gate. The error record is in the format of `batch_eway_error.dtd` (See [Error Messages](#) on page 121). You are required, however, to configure an inbound topic to process this Event.

Required Values

Yes or No. (No is the default).

Include Order Record in Error Record

Description

If this parameter is set to **Yes**, the Batch e*Way includes an Order Record as part of an error record when [Publish Status Record on Error](#) is enabled.

Required Values

Yes or No. (No is the default).

Include Payload in Error Record

Description

If this parameter is set to **Yes**, the Batch e*Way includes the Payload as part of an Error Record when the **Order Record Command** is **Send**.

*Note: This parameter is not used by the PeopleSoft Batch e*Way.*

Required Values

Yes or No. (No is the default).

Action on Malformed Command

Description

If [Enable Message Configuration](#) is set to **Yes**, the Batch e*Way requires a specific XML message structure. This parameter specifies the action that the Batch e*Way takes when the Outgoing Event doesn't match the XML message structure the e*Way requires.

*Note: This parameter is not used by the PeopleSoft Batch e*Way.*

Required Values

One of the following values:

- **Exit** (the default value)
- **Ignore**
- **Raise Alert**
- **Publish Error Record**

API Functions

This chapter describes the most basic Monk functions used by the PeopleSoft Batch e*Way.

7.1 Overview

Two complete sets of Monk functions are used by the PeopleSoft Batch e*Way. This chapter lists only a subset of these functions, which are closely interlinked with [Monk Configuration](#) on page 79, and [Basic e*Way Processes](#) on page 65:

[Batch e*Way Standard Functions](#) on page 94

[ODBC e*Way Standard Functions](#) on page 101

[Generic e*Way Functions](#) on page 113

For descriptions of the entire set of Batch e*Way Monk functions, please see the *Batch e*Way Intelligent Adapter User's Guide*.

For descriptions of the entire set of ODBC e*Way Monk functions, please see the *e*Way Intelligent Adapter for ODBC User's Guide*.

7.2 Batch e*Way Standard Functions

The functions in this category are those called by Batch e*Way configuration parameters (see [Monk Configuration](#) on page 79), and include:

[batch-ack](#) on page 94

[batch-nak](#) on page 97

[batch-exchange-data](#) on page 95

[batch-proc-out](#) on page 98

[batch-ext-connect](#) on page 95

[batch-regular-proc-out](#) on page 98

[batch-ext-shutdown](#) on page 96

[batch-shutdown-notify](#) on page 99

[batch-ext-verify](#) on page 96

[batch-startup](#) on page 100

[batch-init](#) on page 97

batch-ack

Description

Acknowledgment function that is called automatically when the e*Way successfully processes and queues Events from the external system.

Signature

(batch-ack *command*)

Parameters

Name	Type	Description
command	String	Any non-null string

Returns

Returns the string "FAILURE" on all errors; otherwise, returns a *null string*.

Throws

None.

Location

batch-ack.monk

Additional Information

This function is called only 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 a Collaboration for further processing. If the Event's processing is completed successfully, the e*Way executes the [Positive Acknowledgment Function](#); otherwise, the e*Way executes the [Negative Acknowledgment Function](#).

This function can return an Event to be queued, but the e*Way does not return a positive or negative acknowledgement to the external system.

The e*Way exits if it fails its attempt to invoke this function or this function returns a "FAILURE" string.

batch-exchange-data

Description

Initiates an exchange of Events (either inbound or outbound) with an external system.

Signature

(batch-exchange-data)

Parameters

None.

Returns

Returns a *null string* if the function processed an *outbound* Event successfully; otherwise, returns a string to be packaged as an *inbound* Event.

Throws

None.

Location

batch-exchange-data.monk

batch-ext-connect

Description

Establishes or re-establishes a connection to the external system.

Signature

(batch-ext-connect)

Parameters

None.

Returns

Returns the string "UP" if the connection was made successfully; otherwise, returns the string "DOWN".

Throws

except-method

Location

batch-ext-connect.monk

batch-ext-shutdown

Description

Shuts down the connection between the external system and the e*Way.

Signature

(batch-ext-shutdown *command*)

Parameters

Name	Type	Description
command	String	Any non-null string

Returns

Returns a *null string*.

Throws

except-method

Location

batch-ext-shutdown.monk

batch-ext-verify

Description

Confirms that the external system is operating and available.

Signature

(batch-ext-verify)

Parameters

None.

Returns

Returns the string "UP" if the connection was verified successfully; otherwise, returns the string "DOWN".

Throws

except-method

Location

batch-ext-verify.monk

batch-init

Description

Loads the library file **stc_monkfilesys.dll**, and defines exceptions and variables upon which other e*Way functions depend.

Signature

(batch-init)

Parameters

None.

Returns

Returns the string "FAILURE" on all errors; otherwise, returns a *null string*.

Throws

except-method

Location

batch-init.monk

batch-nak

Description

Called automatically when the e*Way fails to process and queue Events from the external system.

Signature

(batch-nak *command*)

Parameters

Name	Type	Description
command	String	Any non-null string

Returns

Returns the string "FAILURE" on all errors; otherwise, returns a *null string*.

Throws

None.

Location

batch-nak.monk

Additional Information

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 a Collaboration for further processing. If the

Event's processing is completed unsuccessfully, the e*Way executes the **Positive Acknowledgment Function**; otherwise, the e*Way executes the **Negative Acknowledgment Function**.

This function can return an Event to be queued, but the e*Way does not return a positive or negative acknowledgement to the external system.

The e*Way exits if it fails its attempt to invoke this function or this function returns a "FAILURE" string.

batch-proc-out

Description

Sends the outbound Event from the e*Way to the external system.

Signature

(batch-proc-out *Event*)

Parameters

Name	Type	Description
Event	String	The Event to be sent

Returns

Returns one of the following strings:

- ◆ Null ("")
- ◆ "RESEND"
- ◆ "CONNERR"
- ◆ "DATAERR"

Throws

None.

Location

batch-proc-out.monk

batch-regular-proc-out

Description

Sends the outbound Event from the e*Way to the external system.

Signature

(batch-regular-proc-out *Event*)

Parameters

Name	Type	Description
Event	String	The Event to be sent

Returns

Returns one of the following strings:

- ◆ *Null* (“”)
- ◆ “RESEND”
- ◆ “CONNERR”
- ◆ “DATAERR”

Throws

None.

Location

batch-regular-proc-out.monk

batch-shutdown-notify

Description

Notifies the external system that the e*Way is shutting down.

Signature

(*batch-shutdown-notify command*)

Parameters

Name	Type	Description
command	String	Any non-null string

Returns

Returns a *null string*.

Throws

None.

Location

batch-shutdown-notify.monk

batch-startup

Description

Launches a Monk function that starts the e*Way. The function that is invoked depends on whether the e*Way uses FTP or file transfer via **copy**, as selected by the **File Transfer Method** configuration parameter.

Signature

(batch-startup)

Parameters

None.

Returns

Returns the string "FAILURE" on all errors; otherwise, returns a *null string*.

Throws

except-method

Location

batch-startup.monk

7.3 ODBC e*Way Standard Functions

The functions in this category are those called by ODBC e*Way configuration parameters (see [Monk Configuration](#) on page 79), and include:

[db-stdver-conn-estab](#) on page 101

[db-stdver-conn-shutdown](#) on page 103

[db-stdver-conn-ver](#) on page 103

[db-stdver-data-exchg](#) on page 105

[db-stdver-data-exchg-stub](#) on page 105

[db-stdver-init](#) on page 106

[db-stdver-neg-ack](#) on page 106

[db-stdver-pos-ack](#) on page 107

[db-stdver-proc-outgoing](#) on page 108

[db-stdver-proc-outgoing-stub](#) on page 109

[db-stdver-shutdown](#) on page 110

[db-stdver-startup](#) on page 111

db-stdver-conn-estab

Description

Used to establish external system connection. The following tasks are performed by this function:

- Construct a new connection handle
- Call db-long to connect to database
- Set up timestamp format, if required
- Set up maximum long data buffer limit, if required
- Bind a dynamic SQL statement and stored procedures

Signature

(db-stdver-conn-estab)

Parameters

None.

Returns

Returns the string "UP" or "SUCCESS" if connection established, any other string if connection not established.

Throws

None.

Additional Information

In order to use the standard database time format, the following function call has been added to this function (immediately before the call to the **db-bind** function):

```
(db-std-timestamp-format connection-handle)
```

To override the use of the standard database time format, the **db-std-timestamp-format** function call should be removed.

For *Maximum Long Data Size* the ODBC library allocates an internal buffer for each **SQL_LONGVARCHAR** and **SQL_LONGVARBINARY** data, when the SQL statement or stored procedure that contains these data types are bound. The default size of each internal data buffer is 1 MB (1048576 bytes). If the user needs to handle long data larger than this default value, add the following function call to specify the maximum data size:

```
(db-max-long-data-size connection-handle maximum-data-size)
```

Examples

```
(define db-stdver-conn-estab
  (lambda ( )
    (let ((result "DOWN")(last_dberr ""))
      (display "[++] Executing e*Way external connection establishment
        function.")
      (display "db-stdver-conn-estab: logging into the database with:\n")
      (display "DATABASE NAME = ")
      (display DATABASE_SETUP_DATABASE_NAME)
      (newline )
      (display "USER NAME = ")
      (display DATABASE_SETUP_USER_NAME)
      (newline )
      (set! connection-handle (make-connection-handle))
      (if (connection-handle? connection-handle)
        (begin
          (if (db-login connection-handle DATABASE_SETUP_DATABASE_NAME
            DATABASE_SETUP_USER_NAME DATABASE_SETUP_ENCRYPTED_PASSWORD)
            (begin
              (db-bind )
              (set! result "UP")
            )
            (begin
              (set! last_dberr (db-get-error-str connection-handle))
              (display last_dberr)
              (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_CANTCONN"
                "ALERTINFO_FATAL" "0" "Cannot connect to database" (string-append
                  "Failed to connect to database: " DATABASE_SETUP_DATABASE_NAME
                  "with error" last_dberr) 0 (list))
              (newline )
              (db-logout connection-handle)
              (set! result "DOWN")
            )
          )
        )
        (begin
          (set! result "DOWN")
          (display "Failed to create connection handle.")
          (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_UNUSABLE"
            "ALERTINFO_FATAL" "0" "database connection handle creation error"
            "Failed to create database connection handle" 0 (list))
          )
        )
        result
      )
    ))
```

db-stdver-conn-shutdown

Description

Called by the system to request that the interface disconnect from the external system, preparing for a suspend/reload cycle.

Signature

(db-stdver-conn-shutdown *string*)

Parameters

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

Returns

Any return value indicates that the suspend can occur immediately, and the interface is placed in the *down* state.

Throws

None.

Examples

```
(define db-stdver-conn-shutdown
  (lambda ( message-string )
    (let ((result "SUCCESS"))
      (comment "Std e*Way connection shutdown function" "[++] Usage:
        Function called by system to request that the interface
        disconnect from the external system, preparing for a suspend/
        reload cycle. Any return value indicates that the suspend can
        occur immediately, and the interface is placed in the down state.
        [++] Input to expect: Function should not expect input. [++]
        Expected return values: anything indicates that the external is
        ready to suspend.n")
      (comment "db-stdver-conn-shutdown [++] Implementation specific
        comment" "none")
      (display "[++] Executing e*Way external connection shutdown
        function.")
      (display message-string)
      (db-logout connection-handle)
      result
    )
  ))
```

db-stdver-conn-ver

Description

Used to verify whether or not the external system connection has been established.

Signature

(db-stdver-conn-ver)

Parameters

None.

Returns

The string "UP" or "SUCCESS" if connection established, any other string if connection not established.

Throws

None.

Additional Information

To use standard database time format, add the following function call to this function: (**db-std-timestamp-format** *connection-handle*) after the (**db-bind**) call.

This SQL statement is designed for database management systems other than Oracle. The use of this function occasionally results in an error in the e*Way's log file. Despite the error, however, the function will complete successfully.

Note: *To users of earlier versions of DART: db-check-connect calls should be replaced with db-alive calls.*

Examples

```
(define db-stdver-conn-ver
  (lambda ( )
    (let ((result "DOWN") (last_dberr ""))
      (display "[++] Executing e*Way external connection verification
function.")
      (display "db-stdver-conn-ver: checking connection status...\n")
      (cond ((string=? STCDB "SYBASE") (db-sql-select connection-handle
"verify" "select getdate()")) ((string=? STCDB "ORACLE8i") (db-
sql-select connection-handle "verify" "select sysdate from
dual")) ((string=? STCDB "ORACLE8") (db-sql-select connection-
handle "verify" "select sysdate from dual")) ((string=? STCDB
"ORACLE7") (db-sql-select connection-handle "verify" "select
sysdate from dual")) (else (db-sql-select connection-handle
"verify" "select {fn NOW()}"))))
      (if (db-alive connection-handle)
        (begin
          (db-sql-fetch-cancel connection-handle "verify")
          (set! result "UP")
        )
        (begin
          (set! last_dberr (db-get-error-str connection-handle))
          (display last_dberr)
          (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_LOSTCONN"
"ALERTINFO_FATAL" "0" "Lost connection to database" (string-
append "Lost connection to database: "
DATABASE_SETUP_DATABASE_NAME "with error" last_dberr) 0 (list))
          (set! result "DOWN")
        )
      )
      result
    )
  ))
```

db-stdver-data-exchg

Description

Used for sending a received Event from the external system to e*Gate. The function expects no input.

Signature

```
(db-stdver-data-exchg)
```

Parameters

None.

Returns

- A message-string indicates a successful operation, and the Event is sent to e*Gate
- An empty string indicates a successful operation, but nothing is sent to e*Gate
- The string "CONNERR" indicates loss of connection with the external

In the latter case, the client moves to a *down* state and attempts to connect; upon reconnecting, this function is re-executed using the same input message.

Throws

None.

Examples

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

db-stdver-data-exchg-stub

Description

Used as a place holder for the function entry point for sending an Event from the external system to e*Gate. The function expects no input.

Note: This function should **not** be used when the interface is configured as an *outbound-only* connection.

Signature

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

Parameters

None.

Returns

- A message-string indicates a successful operation, and the Event is sent to e*Gate

- An empty string indicates a successful operation, but nothing is sent to e*Gate
- The string "CONNERR" indicates loss of connection with the external

In the latter case, the client moves to a *down* state and attempts to connect; upon reconnecting, this function is re-executed using the same input message.

Throws

None.

Examples

```
(define db-stdver-data-exchg-stub
  (lambda ( )
    (let ((result ""))
      (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_INTEREST"
        "ALERTINFO_NONE" "0" "Possible configuration error." "Default
        away data exchange function called." 0 (list))
      result
    )
  ))
```

db-stdver-init

Description

This function begins the initialization process for the e*Way. It loads all of the Monk extension library files that the other e*Way functions access.

Signature

```
(db-stdver-init)
```

Parameters

None.

Returns

If a "FAILURE" string is returned, the e*Way shuts down; any other return indicates success.

Throws

None.

db-stdver-neg-ack

Description

Used to send a negative acknowledgement to the external system, and for post-processing after failing to send data to e*Gate.

Signature

```
(db-stdver-neg-ack message-string)
```

Parameters

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

Returns

- A *null string* indicates a successful operation.
- The string "CONNERR" indicates a loss of connection with the external

In the latter case, the client moves to a *down* state and attempts to connect; upon reconnection, this function is re-executed.

Throws

None.

Examples

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

db-stdver-pos-ack

Description

Used to send a positive acknowledgement to the external system, and for post-processing after successfully sending data to e*Gate.

Signature

```
(db-stdver-pos-ack message-string)
```

Parameters

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

Returns

- A *null string* indicates a successful operation, and the e*Way is then able to proceed with the next request
- The string "CONNERR" indicates a loss of connection with the external

In the latter case, the client moves to a *down* state and attempts to connect; upon reconnection, this function is re-executed.

Throws

None.

Examples

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

db-stdver-proc-outgoing

Description

Used for sending a received message (Event) from e*Gate to the external system.

Signature

```
(db-stdver-proc-outgoing message-string)
```

Parameters

Name	Type	Description
message-string	string	The Event to be processed.

Returns

One of the following strings (if a string other than the following is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function).

- A *null string* indicates a successful operation
- The string "RESEND" causes the message to be resent immediately

In this case, the e*Way compares the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- ♦ If the number of attempts does not exceed the maximum, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, increment the *resend attempts* counter for that message, then repeat the attempt to send the message
- ♦ If the number of attempts exceeds the maximum, the function returns *false* and rolls back the message to the e*Gate IQ from which it was obtained
- The string "CONNERR" indicates that there is a problem communicating with the external system

In this case, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, then calls the **External Connection Establishment Function**

according to the **Down Timeout** schedule, and rolls back the message (Event) to the IQ from which it was obtained.

- The string "DATAERR" indicates that there is a problem with the message (Event) data itself

In this case, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, then increments its *failed message* counter and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see the **Journal File Name** configuration parameter) the message (Event) is saved to the journal file.

Throws

None.

Examples

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

db-stdver-proc-outgoing-stub

Description

Used as a place holder for the function entry point for sending an Event received from e*Gate to the external system (used to catch configuration problems).

Note: *This function should **not** be used when the interface is configured as an inbound-only connection.*

Signature

(db-stdver-proc-outgoing-stub *message-string*)

Parameters

Name	Type	Description
message-string	string	The Event to be processed.

Returns

One of the following strings (if a string other than the following is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function).

- A *null string* indicates a successful operation
- The string "RESEND" causes the message to be resent immediately

In this case, the e*Way compares the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- ◆ If the number of attempts does not exceed the maximum, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, increment the *resend attempts* counter for that message, then repeat the attempt to send the message
- ◆ If the number of attempts exceeds the maximum, the function returns *false* and rolls back the message to the e*Gate IQ from which it was obtained
- The string "CONNERR" indicates that there is a problem communicating with the external system

In this case, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, then calls the **External Connection Establishment Function** according to the **Down Timeout** schedule, and rolls back the message (Event) to the IQ from which it was obtained.

- The string "DATAERR" indicates that there is a problem with the message (Event) data itself

In this case, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, then increments its *failed message* counter and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see the **Journal File Name** configuration parameter) the message (Event) is saved to the journal file.

Throws

None.

Examples

```
(define db-stdver-proc-outgoing-stub
  (lambda ( message-string )
    (let ((result ""))
      (display "[++] Executing e*Way external process outgoing message
function stub.")
      (display message-string)
      (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_INTEREST"
        "ALERTINFO_NONE" "0" "Possible configuration error." (string-
          append "Default away process outgoing msg function passed
following message: " msg) 0 (list))
      result
    )
  ))
```

db-stdver-shutdown

Description

Called by the system to request that the external shutdown, a return value of **SUCCESS** indicates that the shutdown can occur immediately, any other return value indicates that the shutdown Event must be delayed. You are then required to execute a shutdown-request call from within a monk function to allow the requested shutdown process to continue.

Signature

(db-stdver-shutdown *shutdown_notification*)

Parameters

Name	Type	Description
shutdown_notification	string	When the e*Way calls this function, it passes the string "SHUTDOWN_NOTIFICATION" as the parameter.

Returns

The string "SUCCESS" allows an immediate shutdown to occur; anything else delays shutdown until [shutdown-request](#) is executed successfully.

Throws

None.

Examples

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

db-stdver-startup

Description

Used for instance specific function loads and invokes setup.

Signature

(db-stdver-startup)

Parameters

None.

Returns

The string "FAILURE" causes shutdown of the e*Way; any other return indicates success.

Throws

None.

Examples

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


7.4 Generic e*Way Functions

The functions described in this section control the e*Way's most basic operations, and can only be used by the functions defined within the e*Way's configuration file. None of these functions is available to Collaboration Rules scripts executed by the e*Way.

The current set of basic Monk functions is:

- [event-commit-to-egate](#) on page 113
- [event-rollback-to-egate](#) on page 114
- [event-send-to-egate](#) on page 114
- [event-send-to-egate-ignore-shutdown](#) on page 115
- [event-send-to-egate-no-commit](#) on page 115
- [get-logical-name](#) on page 116
- [insert-exchange-data-event](#) on page 116
- [send-external-up](#) on page 117
- [send-external-down](#) on page 117
- [shutdown-request](#) on page 118
- [start-schedule](#) on page 118
- [stop-schedule](#) on page 119
- [waiting-to-shutdown](#) on page 119

event-commit-to-egate

Description

Commits the Event sent previously to the e*Gate system using [event-send-to-egate-no-commit](#).

Signature

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

Parameters

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

Returns

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

Throws

None.

event-rollback-to-egate

Description

Rolls back the Event sent previously to the e*Gate system using [event-send-to-egate-no-commit](#), following receipt of a rollback command from the external system.

Signature

(event-rollback-to-egate *string*)

Parameters

Name	Type	Description
string	string	The data to be rolled back to the e*Gate system.

Returns

Boolean true (#t) if the data is rolled back successfully; otherwise, false (#f).

Throws

None.

event-send-to-egate

Description

Sends data that the e*Way has already received from the external system into the e*Gate system as an Event.

Signature

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

Parameters

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

Returns

A Boolean true (#t) if the data is sent successfully; otherwise, a Boolean 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.

See also

[event-send-to-egate-ignore-shutdown](#) on page 115

[event-send-to-egate-no-commit](#) on page 115

event-send-to-egate-ignore-shutdown

Description

Sends data that the e*Way has already received from the external system into the e*Gate system as an Event—but ignores any pending shutdown issues.

Signature

(event-send-to-egate-ignore-shutdown *string*)

Parameters

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

Returns

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

Throws

None.

See also

[event-send-to-egate](#) on page 114

[event-send-to-egate-no-commit](#) on page 115

event-send-to-egate-no-commit

Description

Sends data that the e*Way has received from the external system to the e*Gate system as an Event—but without Committing, pending confirmation from the external system of correct transmission of the data.

Signature

(event-send-to-egate-no-commit *string*)

Parameters

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

Returns

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

Throws

None.

See also

[event-commit-to-egate](#) on page 113

[event-rollback-to-egate](#) on page 114

[event-send-to-egate](#) on page 114

[event-send-to-egate-ignore-shutdown](#) on page 115

get-logical-name

Description

Returns the logical name of the e*Way.

Signature

(get-logical-name)

Parameters

None.

Returns

The name of the e*Way (as defined by the e*Gate Schema Designer).

Throws

None.

insert-exchange-data-event

Description

While the [Exchange Data with External Function](#) is still active, this function can be called to initiate a repeat call to it—whether or not data was queued to e*Gate via the function's return mechanism following the initial call.

Signature

(insert-exchange-data-event)

Parameters

None.

Returns

None.

Throws

None.

See also

[Exchange Data Interval](#) on page 75

[Zero Wait Between Successful Exchanges](#) on page 75

send-external-up

Description

Informs the e*Way that the connection to the external system is up.

Signature

(send-external-up)

Parameters

None.

Returns

None.

Throws

None.

send-external-down

Description

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

Signature

(send-external-down)

Parameters

None.

Returns

None.

Throws

None.

shutdown-request

Description

Completes the e*Gate shutdown procedure that was initiated by the Control Broker but was interrupted by returning a non-null value within the **Shutdown Command Notification Function**. Once this function is called, shutdown proceeds immediately.

Signature

(shutdown-request)

Parameters

None.

Returns

None.

Throws

None.

Additional Information

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.

start-schedule

Description

Requests that the e*Way execute the **Exchange Data with External Function** specified within the e*Way's configuration file. Does not affect any defined schedules.

Signature

(start-schedule)

Parameters

None.

Returns

None.

Throws

None.

stop-schedule

Description

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 effect any defined schedules, and does not halt the e*Way process itself.

Signature

(stop-schedule)

Parameters

None.

Returns

None.

Throws

None.

waiting-to-shutdown

Description

Informs the external application that a shutdown command has been issued.

Signature

(waiting-to-shutdown)

Parameters

None.

Returns

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

Throws

None.

Document Type Definitions

This appendix provides Document Type Definitions (DTDs) for the XML Messages used in Dynamic Configuration.

A.1 Receive XML Messages

The DTD below provides details of the XML Message that can be used for Receive orders.

```
<!-- Copyright (C) 2000, SeeBeyond Technology Corporation, All rights reserved. -->
<!-- batch eway order record format. -->
<!ELEMENT batch_eWay_order (command,
                            (order_record)+,
                            payload?) >
<!ELEMENT command (#PCDATA) >
<!ATTLIST command Enumeration (send|receive) "send" >
<!ELEMENT order_record (external_host_setup?,
                        (subscribe_to_external|publish_to_external)?,
                        FTP?,
                        SOCKS?) >
<!ELEMENT external_host_setup (host_type?,
                               external_host_name?,
                               user_name?,
                               encrypted_password?,
                               file_transfer_method?,
                               return_tag?) >
<!ELEMENT host_type (#PCDATA) >
<!ELEMENT external_host_name (#PCDATA) >
<!ELEMENT user_name (#PCDATA) >
<!ELEMENT encrypted_password (#PCDATA) >
<!ELEMENT file_transfer_method (#PCDATA) >
<!ATTLIST file_transfer_method Enumeration (ftp|copy) "ftp" >
<!ELEMENT return_tag (#PCDATA) >
<!ELEMENT subscribe_to_external (remote_directory_name?,
                                 remote_file_regexp?,
                                 remote_command_after_transfer?,
                                 remote_rename_or_archive_name?,
                                 local_command_after_transfer?,
                                 local_archive_directory?) >
<!ELEMENT remote_directory_name (#PCDATA) >
<!ELEMENT remote_file_regexp (#PCDATA) >
<!ELEMENT remote_command_after_transfer (#PCDATA) >
<!ATTLIST remote_command_after_transfer Enumeration
(archive|delete|none|rename) "delete" >
<!ELEMENT remote_rename_or_archive_name (#PCDATA) >
<!ELEMENT local_command_after_transfer (#PCDATA) >
```



```

<!ATTLIST local_command_after_transfer Enumeration (archive|delete)
"delete" >
<!ELEMENT local_archive_directory (#PCDATA) >
<!ELEMENT publish_to_external (remote_directory_name?,
remote_file_name?,
append_or_overwrite_when_transferring_files?,
remote_command_after_transfer?,
remote_rename_or_archive_name?,
local_command_after_transfer?,
local_archive_directory?) >
<!ELEMENT remote_file_name (#PCDATA) >
<!ELEMENT append_or_overwrite_when_transferring_files (#PCDATA) >
<!ATTLIST append_or_overwrite_when_transferring_files Enumeration
(append|overwrite) "append" >
<!ELEMENT FTP (server_port,
mode) >
<!ELEMENT server_port (#PCDATA) >
<!ELEMENT mode (#PCDATA) >
<!ELEMENT SOCKS
(server_host_name, server_port, method, user_name, encrypted_password) >
<!ELEMENT server_host_name (#PCDATA) >
<!ELEMENT method (#PCDATA) >
<!ELEMENT payload (#PCDATA) >

```

A.2 Error Messages

The DTD below is used for the Error Reporting XML Message.

```

<!-- Copyright (C) 2000, SeeBeyond Technology Corporation, All rights
reserved. -->
<!-- batch away error record format. -->
<!ELEMENT batch_eWay_error (command,
(return_tag|order_record)?,
error_record,
payload?) >
<!ELEMENT command (#PCDATA) >
<!ATTLIST command Enumeration (send|receive) "send" >
<!ELEMENT order_record (external_host_setup?,
(subscribe_to_external|publish_to_external)?,
FTP?,
SOCKS?) >
<!ELEMENT external_host_setup (host_type?,
external_host_name?,
user_name?,
encrypted_password?,
file_transfer_method?,
return_tag?) >
<!ELEMENT host_type (#PCDATA) >
<!ELEMENT external_host_name (#PCDATA) >
<!ELEMENT user_name (#PCDATA) >
<!ELEMENT encrypted_password (#PCDATA) >
<!ELEMENT file_transfer_method (#PCDATA) >
<!ATTLIST file_transfer_method Enumeration (ftp|copy) "ftp" >
<!ELEMENT return_tag (#PCDATA) >
<!ELEMENT subscribe_to_external (remote_directory_name?,
remote_file_regexp?,
remote_command_after_transfer?,
remote_rename_or_archive_name?,

```

```

                                local_command_after_transfer?,
                                local_archive_directory?) >
<!ELEMENT remote_directory_name (#PCDATA) >
<!ELEMENT remote_file_regexp (#PCDATA) >
<!ELEMENT remote_command_after_transfer (#PCDATA) >
<!ATTLIST remote_command_after_transfer Enumeration
(archive|delete|none|rename) "delete" >
<!ELEMENT remote_rename_or_archive_name (#PCDATA) >
<!ELEMENT local_command_after_transfer (#PCDATA) >
<!ATTLIST local_command_after_transfer Enumeration (archive|delete)
"delete" >
<!ELEMENT local_archive_directory (#PCDATA) >
<!ELEMENT publish_to_external (remote_directory_name?,
                                remote_file_name?,
                                append_or_overwrite_when_transferring_files?,
                                remote_command_after_transfer?,
                                remote_rename_or_archive_name?,
                                local_command_after_transfer?,
                                local_archive_directory?) >
<!ELEMENT remote_file_name (#PCDATA) >
<!ELEMENT append_or_overwrite_when_transferring_files (#PCDATA) >
<!ATTLIST append_or_overwrite_when_transferring_files Enumeration
(append|overwrite) "append" >
<!ELEMENT FTP (server_port,
                                mode) >
<!ELEMENT server_port (#PCDATA) >
<!ELEMENT mode (#PCDATA) >
<!ELEMENT SOCKS
(server_host_name, server_port, method, user_name, encrypted_password) >
<!ELEMENT server_host_name (#PCDATA) >
<!ELEMENT method (#PCDATA) >
<!ELEMENT payload (#PCDATA) >
<!ELEMENT error_record (error_code,
                                error_text,
                                last_action) >
<!ELEMENT error_code (#PCDATA) >
<!ELEMENT error_text (#PCDATA) >
<!ELEMENT last_action (#PCDATA) >

```

A.3 Data Message

The DTD file below provides a data structure, includes a data payload, and is used for transporting data to a Batch e*Way.

```

<!-- Copyright (C) 2000, SeeBeyond Technology Corporation, All rights
reserved. -->
<!-- batch_eway_data record format. -->
<!ELEMENT batch_eway_data (command,
                                (return_tag|order_record)?,
                                payload) >
<!ELEMENT command (#PCDATA) >
<!ATTLIST command Enumeration (send|receive) "send" >
<!ELEMENT order_record (external_host_setup?,
                                (subscribe_to_external|publish_to_external)?,
                                FTP?,
                                SOCKS?) >
<!ELEMENT external_host_setup (host_type?,
                                external_host_name?,

```

```

        user_name?,
        encrypted_password?,
        file_transfer_method?,
        return_tag?) >
<!ELEMENT host_type (#PCDATA) >
<!ELEMENT external_host_name (#PCDATA) >
<!ELEMENT user_name (#PCDATA) >
<!ELEMENT encrypted_password (#PCDATA) >
<!ELEMENT file_transfer_method (#PCDATA) >
<!ATTLIST file_transfer_method Enumeration (ftp|copy) "ftp" >
<!ELEMENT return_tag (#PCDATA) >
<!ELEMENT subscribe_to_external (remote_directory_name?,
        remote_file_regexp?,
        remote_command_after_transfer?,
        remote_rename_or_archive_name?,
        local_command_after_transfer?,
        local_archive_directory?) >
<!ELEMENT remote_directory_name (#PCDATA) >
<!ELEMENT remote_file_regexp (#PCDATA) >
<!ELEMENT remote_command_after_transfer (#PCDATA) >
<!ATTLIST remote_command_after_transfer Enumeration
(archive|delete|none|rename) "delete" >
<!ELEMENT remote_rename_or_archive_name (#PCDATA) >
<!ELEMENT local_command_after_transfer (#PCDATA) >
<!ATTLIST local_command_after_transfer Enumeration (archive|delete)
"delete" >
<!ELEMENT local_archive_directory (#PCDATA) >
<!ELEMENT publish_to_external (remote_directory_name?,
        remote_file_name?,

        append_or_overwrite_when_transferring_files?,
        remote_command_after_transfer?,
        remote_rename_or_archive_name?,
        local_command_after_transfer?,
        local_archive_directory?) >
<!ELEMENT remote_file_name (#PCDATA) >
<!ELEMENT append_or_overwrite_when_transferring_files (#PCDATA) >
<!ATTLIST append_or_overwrite_when_transferring_files Enumeration
(append|overwrite) "append" >
<!ELEMENT FTP (server_port,
        mode) >
<!ELEMENT server_port (#PCDATA) >
<!ELEMENT mode (#PCDATA) >
<!ELEMENT SOCKS
(server_host_name, server_port, method, user_name, encrypted_password) >
<!ELEMENT server_host_name (#PCDATA) >
<!ELEMENT method (#PCDATA) >
<!ELEMENT payload (#PCDATA) >

```

ODBC Monk Scripts

7.5 Sample Monk Scripts

This section includes sample Monk scripts that demonstrate how to use the ODBC e*Way's Monk functions to implement the following activities:

[Initializing Monk Extensions](#) on page 125

[Calling Stored Procedures](#) on page 126

[Inserting Records with Dynamic SQL Statements](#) on page 128

[Updating Records with Dynamic SQL Statements](#) on page 130

[Selecting Records with Dynamic SQL Statements](#) on page 132

[Deleting Records with Dynamic SQL Statements](#) on page 134

[Inserting a Binary Image to a Database](#) on page 135

[Retrieving an Image from a Database](#) on page 138

[Common Supporting Routines](#) on page 140

7.5.1 Initializing Monk Extensions

The sample script shows how to initialize the Monk extensions. This function is used by many of the other sample Monk scripts shown in this chapter.

To use this sample script in an actual implementation, modify the following values:

- **EGATE** – This designates the location of the e*Gate client.
- **dsn** – This is the name of the data source.
- **uid** – This is the user name.
- **pwd** – This is the login password.

```
;demo-init.monk

(define EGATE "/eGate/client")

; routine to load DART Monk extension
(define (load-library extension)
  (define filename (string-append EGATE "/bin/" extension))
  (if (file-exists? filename)
      (load-extension filename)
      (begin
        (display (string-append "File " filename " does not
exist.\n"))
        (abort filename)
      )
  )
)

(load-library "stc_monkext.dll")

;;
;; define STCDB variables, data source, user ID, and password
;;

(define STCDB "ORACLE8")

(load-library "stc_dbmonkext.dll")

(define dsn "database")
(define uid "Administrator")
(define pwd (encrypt-password uid "password"))
```

7.5.2 Calling Stored Procedures

This script gives an example of calling Stored Procedures (see the *e*Way Intelligent Adapter for ODBC User's Guide*) Refer to [Benefits of Stored Procedures](#) on page 60 for additional details.

```

;demo-proc-execute.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; call stored procedure and display results
(define (execute-procedure hdbc hstmt)
  (let ((prm-count (db-proc-param-count hdbc hstmt)))
    (if (db-proc-execute hdbc hstmt)
        (begin
          (do ((col-count (db-proc-column-count hdbc hstmt) (db-
proc-column-count hdbc hstmt)))
              ((or (not (number? col-count)) (= col-count 0)))
              (display-proc-column-property hdbc hstmt col-count)
              (display-proc-column-value hdbc hstmt col-count)
            )
          (display-proc-parameter-output-value hdbc hstmt prm-count)
          (if (db-proc-return-exist hdbc hstmt)
              (begin
                (display "return: value = ")
                (display (db-proc-return-value hdbc hstmt))
                (newline)
              )
            )
          (display (db-get-error-str hdbc))
        )
      )
  )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
  )

(if (db-login hdbc dsn uid pwd)
    (begin
      (display "\ndatabase login succeed !\n")

      ; bind the stored procedure
      (define hstmt1 (bind-procedure hdbc "PERSONNEL.GET_EMPLOYEES"))

      ; call the stored procedure if the binding is successful
      (if (statement-handle? hstmt1)
          (begin
            (display "call PERSONNEL.GET_EMPLOYEES to get all sales
...\n\n")
            (if (and
                  (db-proc-param-assign hdbc hstmt1 0 "30")
                  (db-proc-param-assign hdbc hstmt1 1 "10")
                )
                (execute-procedure hdbc hstmt1)
                (display (db-get-error-str hdbc))
              )
          )
        )
  )
)

```

```
    )  
    (if (not (db-logout hdbc))  
        (display (db-get-error-str hdbc))  
    )  
    )  
    (display (db-get-error-str hdbc))  
    )
```

7.5.3 Inserting Records with Dynamic SQL Statements

```
;demo-stmt-insert.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
  (if (db-stmt-execute hdbc hstmt)
      (begin
        (display-stmt-row-count hdbc hstmt)
        #t
      )
      #f
  )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)

(define stmt1 "INSERT INTO SCOTT.BONUS SELECT ENAME, JOB, SAL, COMM
FROM SCOTT.EMP WHERE DEPTNO = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
          (begin
            (display "\nInsert accounting department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "10")
                (if (execute-statement hdbc hstmt1)
                    (begin
                      (display "\nCommit the insertions ...\n")
                      (if (not (db-commit hdbc))
                          (display (db-get-error-str hdbc))
                      )
                    )
                    (display (db-get-error-str hdbc))
                )
                (display (db-get-error-str hdbc))
            )
            (display "\nInsert sales department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "20")
                (if (execute-statement hdbc hstmt1)
                    (begin
                      (display "\nCommit the insertions ...\n")
                      (if (not (db-commit hdbc))
                          (display (db-get-error-str hdbc))
                      )
                    )
                    (display (db-get-error-str hdbc))
                )
            )
          )
      )
    )
)
```



```
        )  
        (display (db-get-error-str hdbc))  
    )  
    )  
    (if (not (db-logout hdbc))  
        (display (db-get-error-str hdbc))  
    )  
    )  
    (display (db-get-error-str hdbc))  
    )  
    )
```

7.5.4 Updating Records with Dynamic SQL Statements

```
;demo-stmt-update.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
  (if (db-stmt-execute hdbc hstmt)
      (begin
        (display-stmt-row-count hdbc hstmt)
        #t
      )
      #f
  )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)

(define stmt1 "UPDATE SCOTT.BONUS SET COMM = ? WHERE JOB = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
          (begin
            (display "\nUpdate commission of manager ...\n")
            (if
              (and
                (db-stmt-param-assign hdbc hstmt1 0 "10")
                (db-stmt-param-assign hdbc hstmt1 1 "MANAGER")
              )
              (if (execute-statement hdbc hstmt1)
                  (begin
                    (display "\nCommit the updates ...\n")
                    (if (not (db-commit hdbc))
                        (display (db-get-error-str hdbc))
                    )
                  )
                  (display (db-get-error-str hdbc))
                )
              (display (db-get-error-str hdbc))
            )
          )
          (display "\nUpdate commission of clerk ...\n")
          (if
            (and
              (db-stmt-param-assign hdbc hstmt1 0 "20")
              (db-stmt-param-assign hdbc hstmt1 1 "CLERK")
            )
            (if (execute-statement hdbc hstmt1)
                (begin
                  (display "\nCommit the updates ...\n")
                )
            )
          )
        )
    )
)
```


7.5.5 Selecting Records with Dynamic SQL Statements

```
;demo-stmt-select.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
  (if (db-stmt-execute hdbc hstmt)
      (begin
        (display-stmt-column-value hdbc hstmt)
        #t
      )
      #f
  )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)

(define stmt1 "SELECT EMPNO, ENAME, JOB FROM SCOTT.EMP WHERE JOB = ?")
(define stmt2 "SELECT ENAME, DNAME, JOB, HIREDATE FROM SCOTT.EMP,
SCOTT.DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO AND DEPT.DNAME = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statements
      (define hstmt1 (bind-statement hdbc stmt1))
      (define hstmt2 (bind-statement hdbc stmt2))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
          (begin
            (display "\nList all salesman ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "SALESMAN")
                (if (not (execute-statement hdbc hstmt1))
                    (display (db-get-error-str hdbc))
                )
                (display (db-get-error-str hdbc))
            )
            (display "\nList all manager ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "MANAGER")
                (if (not (execute-statement hdbc hstmt1))
                    (display (db-get-error-str hdbc))
                )
                (display (db-get-error-str hdbc))
            )
          )
      )

      (if (statement-handle? hstmt2)
          (begin
            (display "\nList employee of accounting department
...\n\n")
            (if (db-stmt-param-assign hdbc hstmt2 0 "ACCOUNTING")
                (if (not (execute-statement hdbc hstmt2))
                    (display (db-get-error-str hdbc))
                )
            )
          )
      )
    )
)
```

```
        )  
        (display (db-get-error-str hdbc))  
    )  
    (display (db-get-error-str hdbc))  
)  
(if (not (db-logout hdbc))  
    (display (db-get-error-str hdbc))  
    )  
)  
(display (db-get-error-str hdbc))  
)
```

7.5.6 Deleting Records with Dynamic SQL Statements

```
;demo-stmt-delete.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
  (if (db-stmt-execute hdbc hstmt)
      (begin
        (display-stmt-row-count hdbc hstmt)
        #t
      )
      #f
  )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)

(define stmt1 "DELETE FROM SCOTT.BONUS WHERE ENAME IS NOT NULL")

(if (db-login hdbc dsn uid pwd)
    (begin
      (display "\ndatabase login succeed !\n")

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
          (begin
            (display "\nDelete records from scott.bonus table ...\n")
            (if (execute-statement hdbc hstmt1)
                (begin
                  (display "\nCommit the deletions ...\n")
                  (if (not (db-commit hdbc))
                      (display (db-get-error-str hdbc))
                  )
                )
            (display (db-get-error-str hdbc))
          )
          )
      )
      (if (not (db-logout hdbc))
          (display (db-get-error-str hdbc))
      )
    )
    (display (db-get-error-str hdbc))
)
)
```

7.5.7 Inserting a Binary Image to a Database

This sample shows how to insert a Binary Image into a Database. It uses both Static and Dynamic SQL functions. See the *e*Way Intelligent Adapter for ODBC User's Guide* for more details.

```

;demo-image-insert.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

(define (query-exist hdbc hstmt id)
  (let ((rec-count 0) (result '#()))
    (if (db-stmt-param-assign hdbc hstmt 0 id)
        (if (db-stmt-execute hdbc hstmt)
            (begin
              (set! result (vector-ref (db-stmt-fetch hdbc hstmt) 0))
              (set! rec-count (string->number result))
              (set! result (db-stmt-fetch-cancel hdbc hstmt))
              (if (> rec-count 0)
                  (begin
                    (display "image already exist\n")
                    #t
                  )
                  #f
                )
            )
        (begin
          (display (db-get-error-str hdbc))
          #f
        )
      )
    (begin
      (display (db-get-error-str hdbc))
      #f
    )
  )
)

(define (execute-statement hdbc hstmt)
  (let ((col-count (db-stmt-column-count hdbc hstmt)) (row-count 0))
    (if (db-stmt-execute hdbc hstmt)
        (begin
          (if (> col-count 0)
              (if (not (display-stmt-column-value hdbc hstmt col-
count))
                  (display (db-get-error-str hdbc))
              )
          )
          (set! row-count (db-stmt-row-count hdbc hstmt))
          (if (boolean? row-count)
              (display (db-get-error-str hdbc))
              (display (string-append "number of image insert = "
(number->string row-count) "\n"))
            )
          (newline)
          #t
        )
        #f
      )
  )
)

```

```

)

(define (bind-image-statement hdbc stmt)
  (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
    (display (string-append "\nDynamic statement : " stmt "\n"))
    (if (statement-handle? hstmt)
        (begin
          ; (db-stmt-param-bind hdbc hstmt 0 "SQL_INTEGER" 4 0)
          ; (db-stmt-param-bind hdbc hstmt 1 "SQL_VARCHAR" 20 0)
          ; (db-stmt-param-bind hdbc hstmt 2 "SQL_VARCHAR" 10 0)
          ; (db-stmt-param-bind hdbc hstmt 3 "SQL_INTEGER" 38 0)
          ; (db-stmt-param-bind hdbc hstmt 4 "SQL_INTEGER" 38 0)
          ; (db-stmt-param-bind hdbc hstmt 5 "SQL_INTEGER" 10 0)
          (db-stmt-param-bind hdbc hstmt 6 "SQL_LONGVARIABLE"
2000000 0)
          (define prm-count (db-stmt-param-count hdbc hstmt))
          (display-stmt-parameter-property hdbc hstmt prm-count)

          (define col-count (db-stmt-column-count hdbc hstmt))
          (display-stmt-column-property hdbc hstmt col-count)
        )
        (display (db-get-error-str hdbc)))
    )
  hstmt
)

(define image1-id "7100")
(define image1-name "Coast")
(define image1-type "JPEG")
(define image1-width "1280")
(define image1-height "1024")
(define image1-file (string-append image1-name ".jpg"))

(define image-port (open-input-file image1-file))
(define image1-data (read image-port 1000000))
(close-port image-port)
(define image1-size (number->string (string-length image1-data)))

(define image2-id "7200")
(define image2-name "Launch")
(define image2-type "JPEG")
(define image2-width "2000")
(define image2-height "1600")
(define image2-file (string-append image2-name ".jpg"))

(define image-port (open-input-file image2-file))
(define image2-data (read image-port 2000000))
(close-port image-port)
(define image2-size (number->string (string-length image2-data)))

(define hdbc (make-connection-handle))
(display (connection-handle? hdbc)) (newline)

(define stmt0 "select count(0) from SCOTT.IMAGE where PIX_ID = ?")
(define stmt1 "insert into SCOTT.IMAGE (PIX_ID, PIX_NAME, PIX_TYPE,
BYTE_SIZE, PIX_WIDTH, PIX_HEIGHT, PIX_DATA) values (?, ?, ?, ?, ?, ?,
?)")

(if (db-login hdbc dsn uid pwd)
    (begin
      (display "\ndatabase login succeed !\n")
      (display (db-dbms hdbc)) (newline)
    )
  )

```



```

(display (db-std-timestamp-format hdbc)) (newline)
(display (db-max-long-data-size hdbc 2000000)) (newline)

; bind the query and insert statement
(define hquery (bind-statement hdbc stmt0))
(define hinsert (bind-image-statement hdbc stmt1))

(if (and
    (statement-handle? hquery)
    (statement-handle? hinsert)
)
    (begin
        (if (not (query-exist hdbc hquery image1-id))
            (begin
                (display (string-append "insert image " image1-file "\n"))
                (if (and
                    (db-stmt-param-assign hdbc hinsert 0 image1-id)
                    (db-stmt-param-assign hdbc hinsert 1 image1-name)
                    (db-stmt-param-assign hdbc hinsert 2 image1-type)
                    (db-stmt-param-assign hdbc hinsert 3 image1-size)
                    (db-stmt-param-assign hdbc hinsert 4 image1-width)
                    (db-stmt-param-assign hdbc hinsert 5 image1-height)
                    (db-stmt-param-assign hdbc hinsert 6 image1-data)
                )
                    (if (execute-statement hdbc hinsert)
                        (db-commit hdbc)
                        (display (db-get-error-str hdbc)))
                    (display (db-get-error-str hdbc)))
                )
            )
        )
    )

    (if (not (query-exist hdbc hquery image2-id))
        (begin
            (display (string-append "insert image " image2-file "\n"))
            (if (and
                (db-stmt-param-assign hdbc hinsert 0 image2-id)
                (db-stmt-param-assign hdbc hinsert 1 image2-name)
                (db-stmt-param-assign hdbc hinsert 2 image2-type)
                (db-stmt-param-assign hdbc hinsert 3 image2-size)
                (db-stmt-param-assign hdbc hinsert 4 image2-width)
                (db-stmt-param-assign hdbc hinsert 5 image2-height)
                (db-stmt-param-assign hdbc hinsert 6 image2-data)
            )
                (if (execute-statement hdbc hinsert)
                    (db-commit hdbc)
                    (display (db-get-error-str hdbc)))
                (display (db-get-error-str hdbc)))
            )
        )
    )

    (if (not (db-logout hdbc))
        (display (db-get-error-str hdbc))
    )
)
(display (db-get-error-str hdbc))
)

```

7.5.8 Retrieving an Image from a Database

This sample shows how to Retrieve an image from a Database. It uses both Static and Dynamic SQL functions. See the *e*Way Intelligent Adapter for ODBC User's Guide* for more details.

```
;demo-image-select.monk

; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

(define (get-image hdbc hstmt)
  (do (
    (result (db-stmt-fetch hdbc hstmt) (db-stmt-fetch hdbc
hstmt))
      (first_name "")
      (file_type "")
      (file_name "")
      (width "")
      (height "")
      (output_port '())
    )
    ((boolean? result) result)
    (set! first_name (vector-ref result 0))
    (set! file_type (strip-trailing-whitespace (vector-ref result
1)))
    (set! width (strip-trailing-whitespace (vector-ref result 2)))
    (set! height (strip-trailing-whitespace (vector-ref result 3)))
    (cond
      ((string=? file_type "JPEG") (set! file_name (string-append
first_name ".jpg")))
      ((string=? file_type "GIF") (set! file_name (string-append
first_name ".gif")))
      ((string=? file_type "BITMAP") (set! file_name (string-append
first_name ".bmp")))
      ((string=? file_type "TIFF") (set! file_name (string-append
first_name ".tif")))
      (else (set! file_name (string-append first_name ".raw")))
    )
    (if (file-exists? file_name)
      (file-delete file_name)
    )
    (display (string-append "picture name = " file_name "\n"))
    (display (string-append "picture size = " width " x " height
"\n\n"))
    (set! output_port (open-output-file file_name))
    (display (vector-ref result 4) output_port)
    (close-port output_port)
  )
  )

(define (execute-statement hdbc hstmt)
  (let ((col-count (db-stmt-column-count hdbc hstmt)) (row-count 0))
    (if (db-stmt-execute hdbc hstmt)
      (begin
        (if (> col-count 0)
          (if (not (get-image hdbc hstmt))
            (display (db-get-error-str hdbc))
          )
        )
        (set! row-count (db-stmt-row-count hdbc hstmt))
        (if (boolean? row-count)

```

```

                (display (db-get-error-str hdbc))
                (display (string-append "number of image retrieved = "
(number->string row-count) "\n"))
            )
            (newline)
            #t
        )
        #f
    )
)

(define hdbc (make-connection-handle))
(display (connection-handle? hdbc)) (newline)

(define stmt "select PIX_NAME, PIX_TYPE, PIX_WIDTH, PIX_HEIGHT,
PIX_DATA from SCOTT.IMAGE where PIX_ID = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
        (display "\ndatabase login succeed !\n")
        (display (db-dbms hdbc)) (newline)
        (display (db-std-timestamp-format hdbc)) (newline)
        (display (db-max-long-data-size hdbc 2000000)) (newline)

        ; bind the select statement
        (define hselect (bind-binary-statement hdbc stmt))

        ; execute the dynamic statement
        (display "select IMAGE table\n")
        (if (statement-handle? hselect)
            (begin
                (if (db-stmt-param-assign hdbc hselect 0 "7100")
                    (if (not (execute-statement hdbc hselect))
                        (display (db-get-error-str hdbc))
                    )
                (display (db-get-error-str hdbc))
                )
                (if (db-stmt-param-assign hdbc hselect 0 "7200")
                    (if (not (execute-statement hdbc hselect))
                        (display (db-get-error-str hdbc))
                    )
                (display (db-get-error-str hdbc))
                )
            )
        )
        (if (not (db-logout hdbc))
            (display (db-get-error-str hdbc))
        )
    )
    (display (db-get-error-str hdbc))
)
)

```

7.5.9 Common Supporting Routines

This sample script displays and defines values and parameters for stored procedures. The routines contained in this script are used by many of the Monk samples in this chapter. For more details about functions used in this script, see the *e*Way Intelligent Adapter for ODBC User's Guide* for more details.

```

;demo-common.monk

;;
;; stored procedure auxiliary functions
;;

; display parameter properties of the stored procedure
(define (display-proc-parameter-property hdbc hstmt prm-count)
  (display "parameter count = ") (display prm-count) (newline)
  (do ((i 0 (+ i 1))) ((= i prm-count))
    (display "parameter ")
    (display (db-proc-param-name hdbc hstmt i))
    (display ": type = ")
    (display (db-proc-param-type hdbc hstmt i))
    (display ", io = ")
    (display (db-proc-param-io hdbc hstmt i))
    (newline)
  )
)

; display value of output parameters from stored procedure
(define (display-proc-parameter-output-value hdbc hstmt prm-count)
  (do ((i 0 (+ i 1))) ((= i prm-count))
    (if (not (equal? (db-proc-param-io hdbc hstmt i) "IN"))
      (begin
        (display "output parameter ")
        (display (db-proc-param-name hdbc hstmt i))
        (display " = ")
        (display (db-proc-param-value hdbc hstmt i))
        (newline)
      )
    )
  )
)

; display column properties of the return result set
(define (display-proc-column-property hdbc hstmt col-count)
  (display "column count = ") (display col-count) (newline)
  (do ((i 0 (+ i 1))) ((= i col-count))
    (display "column ")
    (display (db-proc-column-name hdbc hstmt i))
    (display ": type = ")
    (display (db-proc-column-type hdbc hstmt i))
    (newline)
  )
  (newline)
)

; display column value of the return result set of the stored
procedure
(define (display-proc-column-value hdbc hstmt col-count)
  (define (fetch-next)
    (let ((result (db-proc-fetch hdbc hstmt)))
      (if (boolean? result)
          result
          (begin (display result) (newline) (fetch-next)))
    )
  )
)

```

```

        )
    )
    (fetch-next)
    (newline)
)

; bind stored procedure and display parameter properties
(define (bind-procedure hdbc proc)
  (let ((hstmt (db-proc-bind hdbc proc)))
    (if (statement-handle? hstmt)
        (begin
          (display (string-append "bind stored procedure : " proc
"\n"))
          (define prm-count (db-proc-param-count hdbc hstmt))
          (display-proc-parameter-property hdbc hstmt prm-count)
          (newline)
          (if (db-proc-return-exist hdbc hstmt)
              (begin
                (display "return: type = ")
                (display (db-proc-return-type hdbc hstmt))
                (newline)
              )
            )
          (newline)
        )
      (display (db-get-error-str hdbc))
    )
    hstmt
  )
)

;;
;; dynamic statement auxiliary functions
;;

; display parameter properties of the SQL statement
(define (display-stmt-parameter-property hdbc hstmt prm-count)
  (display "parameter count = ") (display prm-count) (newline)
  (do ((i 0 (+ i 1))) ((= i prm-count))
    (display "parameter #")
    (display i)
    (display ": type = ")
    (display (db-stmt-param-type hdbc hstmt i))
    (newline)
  )
  (newline)
)

; display column properties of the SQL statement
(define (display-stmt-column-property hdbc hstmt col-count)
  (display "column count = ") (display col-count) (newline)
  (do ((i 0 (+ i 1))) ((= i col-count))
    (display "column ")
    (display (db-stmt-column-name hdbc hstmt i))
    (display ": type = ")
    (display (db-stmt-column-type hdbc hstmt i))
    (newline)
  )
  (newline)
)

```

```

; display column value of the return result set of the SQL statement
(define (display-stmt-column-value hdbc hstmt)
  (define (fetch-next)
    (let ((result (db-stmt-fetch hdbc hstmt)))
      (if (boolean? result)
          result
          (begin (display result) (newline) (fetch-next)))
      )
    )
  (fetch-next)
  (newline)
)

; display row count affected by the execution of the SQL statement
(define (display-stmt-row-count hdbc hstmt)
  (let ((row-count (db-stmt-row-count hdbc hstmt)))
    (cond
      ((= row-count 0) (display "\n(no row affected)\n"))
      ((= row-count 1) (display "\n(1 row affected)\n"))
      (else (display (string-append "\n(" (number->string row-
count) " rows affected)\n")))
    )
  )
)

; bind dynamic statement and display paramters and column properties
(define (bind-statement hdbc stmt)
  (let ((hstmt (db-stmt-bind hdbc stmt)))
    (display (string-append "\nDynamic statement : " stmt "\n"))
    (if (statement-handle? hstmt)
        (begin
          (define prm-count (db-stmt-param-count hdbc hstmt))
          (display-stmt-parameter-property hdbc hstmt prm-count)

          (define col-count (db-stmt-column-count hdbc hstmt))
          (display-stmt-column-property hdbc hstmt col-count)
        )
        (display (db-get-error-str hdbc)))
    )
  hstmt
)

; bind dynamic statement to input/output raw binary data
(define (bind-binary-statement hdbc stmt)
  (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
    (display (string-append "\nDynamic statement : " stmt "\n"))
    (if (statement-handle? hstmt)
        (begin
          (define prm-count (db-stmt-param-count hdbc hstmt))
          (display-stmt-parameter-property hdbc hstmt prm-count)

          (define col-count (db-stmt-column-count hdbc hstmt))
          (display-stmt-column-property hdbc hstmt col-count)
        )
        (display (db-get-error-str hdbc)))
    )
  hstmt
)

```

Index

A

Action on Malformed Command parameter 91
 Additional Path parameter 79
 Assigning ETDs to Event Types 33
 Autorun 16
 Auxiliary Library Directories parameter 80

B

Base64 88
 Batch e*Way Functions 94–100
 batch mode interface
 operational overview 52
 outbound from PeopleSoft 54
 batch-ack function 94
 batch-exchange-data function 95
 batch-ext-connect function 95
 batch-ext-shutdown function 96
 batch-ext-verify function 96
 batch-init function 97
 batch-nak function 97
 batch-proc-out function 98
 batch-shutdown-notify function 99
 batch-startup function 100
 Build tool 31

C

calling stored procedures, sample 126
 Changing the User Name 45
 Collaboration 34, 49, 50, 64
 Rules 34, 64
 Rules Editor 64
 common supporting routines, sample 140
 Communication Setup 75–78
 Components, e*Way 13
 configuration
 Communication Setup 75–78
 Dynamic Configuration 88–92
 General Settings 73–74
 Monk Configuration 79–87
 configuration parameters
 Action on Malformed Command 91
 Additional Path 79

Auxiliary Library Directories 80
 Down Timeout 77
 Enable Message Configuration 88
 Exchange Data Interval 75
 Exchange Data With External Function 82
 Exchange-if-in-window-on-startup 78
 External Connection Establishment Function 83
 External Connection Shutdown Function 84
 External Connection Verification Function 84
 Forward External Errors 74
 Include Order Record in Error Record 91
 Include Payload in Error Record 91
 Journal File Name 73
 Max Failed Messages 73
 Max Resends Per Message 73
 Monk Environment Initialization File 80
 Negative Acknowledgment Function 86
 Positive Acknowledgement Function 85
 Process Outgoing Message Function 81
 Publish Status Record on Error 91
 Publish Status Record on Success 90
 Resend Timeout 77
 Shutdown Command Notification Function 87
 Start Exchange Data Schedule 76
 Startup Function 81
 Stop Exchange Data Schedule 77
 Up Timeout 77
 Zero Wait Between Successful Exchanges 75
 configuration procedures 41
 continue on error 53
 conventions, writing in document 8
 Creating an e*Way 39

D

db-stdver-conn-estab function 101
 db-stdver-conn-shutdown function 103
 db-stdver-conn-ver function 103
 db-stdver-data-exchg function 105
 db-stdver-data-exchg-stub function 105
 db-stdver-init function 106
 db-stdver-neg-ack function 106
 db-stdver-pos-ack function 107
 db-stdver-proc-outgoing function 108
 db-stdver-proc-outgoing-stub function 109
 db-stdver-shutdown function 110
 db-stdver-startup function 111
 deleting records, sample 134
 Document Type Definition (DTD) 120
 Down Timeout parameter 77
 Dynamic Configuration 88–92

E

- e*Gate to PeopleSoft operation 11, 51
- e*Way
 - Components 13
 - Configuration 41
 - creating 39
 - Installation 16
 - Properties 40
 - Schedules 45
 - Startup Options 45
 - troubleshooting 49
- e*Way Operation
 - e*Gate to PeopleSoft 11, 51
 - PeopleSoft to e*Gate 12, 54
- EBCDIC 15
- Editor
 - Collaboration Rules 64
 - ETD 31
- Enable Message Configuration parameter 88
- error reporting 59
- ETD Editor 31
- Event Type 33
- Event Type Definition (ETD) 33
- event-commit-to-egate function 113
- event-rollback-to-egate function 114
- Events 63
- event-send-to-egate function 114
- event-send-to-egate-ignore-shutdown function 115
- event-send-to-egate-no-commit function 115
- Exchange Data Interval parameter 75
- Exchange Data with External Function parameter 82
- Exchange-if-in-window-on-startup parameter 78
- External Connection Establishment Function parameter 83
- External Connection Shutdown Function parameter 84
- External Connection Verification Function parameter 84

F

- Forward External Errors parameter 74
- functions (see also functions, Monk)
 - Batch e*Way 94–100
 - Generic 113–119
 - ODBC e*Way 101–112
- functions, Monk
 - batch-ack 94
 - batch-exchange-data 95
 - batch-ext-connect 95
 - batch-ext-shutdown 96
 - batch-ext-verify 96
 - batch-init 97

- batch-nak 97
- batch-proc-out 98
- batch-shutdown-notify 99
- batch-startup 100
- db-stdver-conn-estab 101
- db-stdver-conn-shutdown 103
- db-stdver-conn-ver 103
- db-stdver-data-exchg 105
- db-stdver-data-exchg-stub 105
- db-stdver-init 106
- db-stdver-neg-ack 106
- db-stdver-pos-ack 107
- db-stdver-proc-outgoing 108
- db-stdver-proc-outgoing-stub 109
- db-stdver-shutdown 110
- db-stdver-startup 111
- event-commit-to-egate 113
- event-rollback-to-egate 114
- event-send-to-egate 114
- event-send-to-egate-ignore-shutdown 115
- event-send-to-egate-no-commit 115
- get-logical-name 116
- insert-exchange-data-event 116
- send-external down 117
- send-external-up 117
- shutdown-request 118
- start-schedule 118
- stop-schedule 119
- waiting-to-shutdown 119

G

- General Settings 73–74
- Generic e*Way Functions 113–119
- get-logical-name function 116

I

- Include Order Record in Error Record parameter 91
- Include Payload in Error Record parameter 91
- initializing Monk extensions, sample 125
- insert-exchange-data-event function 116
- inserting records, sample 128
- Installation procedure
 - e*Way (UNIX) 21
 - e*Way (Windows) 16
 - sample schema 25
- InstallShield 16
- Intelligent Queue (IQ) 35, 49, 54

J

- Journal File Name parameter 73

L

Load Path, Monk 79
logging options 47

M

Max Failed Messages parameter 73
Max Resends Per Message parameter 73
monitoring thresholds 48
Monk Configuration 79–87
 Load Path 79
 Specifying File Names 79
 Specifying Function Names 79
 Specifying Multiple Directories 79
Monk Environment Initialization File parameter 80

N

Negative Acknowledgment Function parameter 86

O

ODBC e*Way Functions 101–112
operational overview 52, 54
OS/390 15
outbound batch mode interface 54
outbound from PeopleSoft 54

P

parameters - see configuration parameters
Participating Host 49
PeopleSoft to e*Gate operation 12, 54
Positive Acknowledgment Function parameter 85
procedures
 configuration 41
 installation 16
Process Outgoing Message Function parameter 81
processing
 continue on error 53
 rollback on error 53
 stop on error 53
Properties, e*Way 40
Publish Status Record on Error parameter 91
Publish Status Record on Success parameter 90

Q

Queues 35

R

Receiving Data with a Receive Order 57

remote function call (RFC) 13, 54
Resend Timeout parameter 77
rollback on error 53

S

sample
 calling stored procedures 126
 common routines 140
 common supporting routines 140
 deleting records with dynamic SQL statements 134
 dynamic SQL statements 128, 130, 132, 134
 initializing Monk extensions 125
 inserting binary images 135
 inserting records with dynamic SQL statements 128
 retrieving images 138
 selecting records with dynamic SQL statements 132
 stored procedures 126
 updating records with dynamic SQL statements 130
sample schema
 descriptions 35
 installation 25
Schedules 45
selecting records, sample 132
send-external-down function 117
send-external-up function 117
Setting Startup Options or Schedules 45
Shutdown Command Notification Function parameter 87
shutdown-request function 118
SOCKS
 description 15
Start Exchange Data Schedule parameter 76
start-schedule function 118
Startup Function parameter 81
Startup Options 45
Stop Exchange Data Schedule parameter 77
stop on error 53
stop-schedule function 119
stored procedures, sample 126
supported variable SQL datatypes 62

T

troubleshooting the e*Way 49

U

UNIX installation procedure 21

Index

Up Timeout parameter 77
updating records, sample 130
User name 45

W

waiting-to-shutdown function 119
Windows installation procedure 16

X

XML 56

Z

Zero Wait Between Successful Exchanges parameter
75