e*Way Intelligent Adapter for Siebel (Event Driven) User's Guide

Release 4.5.3

SEEBEYOND™

Contents

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:

- Operation and administration of Windows NT or Windows 2000 systems
- Windows-style GUI operations
- Siebel 99 or 2000, and COM/DCOM operations

P.2 Organization

This User's Guide is organized into two parts. The first part, consisting of Chapters 1-4, introduces the e*Way and describes the procedures for installing 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 5-7, describes the architecture and internal functionality of the e*Way. 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 Siebel (Event-Driven) is frequently referred to as the Siebel Event-Driven e*Way, or simply the e*Way.

P.4 Online Viewing

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, or attributes within a function signature, are set within brackets <> as shown below:

```
stcregutil -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.

Introduction

The e*Way Intelligent Adapter for Siebel (Event-Driven) provides connectivity between e*Gate and Siebel 99 Front Office using COM/DCOM connectivity. It provides an inbound and outbound event-driven interface option to or from another system, through e*Gate.

1.1 COM/DCOM Communications

The Microsoft *Component Object Model* (COM) is a component software architecture that allows applications and systems to be built using separate components. COM is the underlying architecture that forms the foundation for higher-level software services. By using COM, software objects can be reused for a variety of applications. Because of its binary standard, COM allows any two components to communicate regardless of the language the components are written in.

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

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

The Siebel Event-Driven e*Way uses an internal DCOM interface to provide connectivity to Siebel's COM Object Manager.

1.2 The Siebel COM Data Server

The Siebel Object Interface method used for implementing the Event-Driven interface is the Siebel COM Data Server. This interfacing method has access to the following Siebel object types:

- Application Object
- Business Component
- Business Object

and apply the Siebel business rules and validations when data is loaded. The following descriptions are based on those given in the Siebel documentation.

1.2.1 Application Object

The Application object represents the Siebel application that is currently active, and is an instance of the Application object type. An Application object is created when a Siebel software application is started. This object contains the properties and events that interact with Siebel software as a whole.

1.2.2 Business Component

A Business Component object is a logical abstraction of one or more database tables, and defines the structure, behavior, and information displayed, of a particular subject such as a product, contact, or account. The information stored in a Business Component is usually specific to a particular subject, and typically is not dependent on other Business Components.

Business Component objects have associated data structured as records, and contain data units called fields. Business Components can be used in one or many Business Objects.

1.2.3 Business Object

Business Objects are modifiable, object-oriented building blocks of Siebel Applications. Business Objects define the relationships between different Business Components, and contain semantic information about items such as sales, marketing, and service-related entities. A Siebel Business Object groups one or more Business Components into a logical unit of information.

Examples of Siebel Business Objects include Opportunity, Quote, Campaign, and Forecast. An Opportunity business object may consist of Opportunity, Contact, and Product Business components, with the Opportunity business component having a parent-child relationship with the other business components.

1.2.4 Object Types

An Object Type is a named structure, and acts as a template from which Object Definitions of that type can be created. An Object Type has a predefined set of properties—Object Definitions created from it have values for each of these properties

1.2.5 Object Definitions

An Object Definition in the Siebel Tools environment implements one piece of the software, such as a user interface, abstract data representation, or direct database representation construct. Items such as a database column, a dialog box, or a join relationship between database tables are implemented as Object Definitions.

An Object Definition consists of properties, which are characteristics of the software construct that is implemented by the Object Definition. For example, the properties of a database column would include its name, data type, and length.

Note: *Siebel Tools object model concepts are* **not** *the same as objects in an object-oriented programming language.*

1.2.6 Object Layers

The object definitions in Siebel Enterprise Applications fall into three separate architectural layers (excluding the third-party DBMS), as shown in Figure 1.
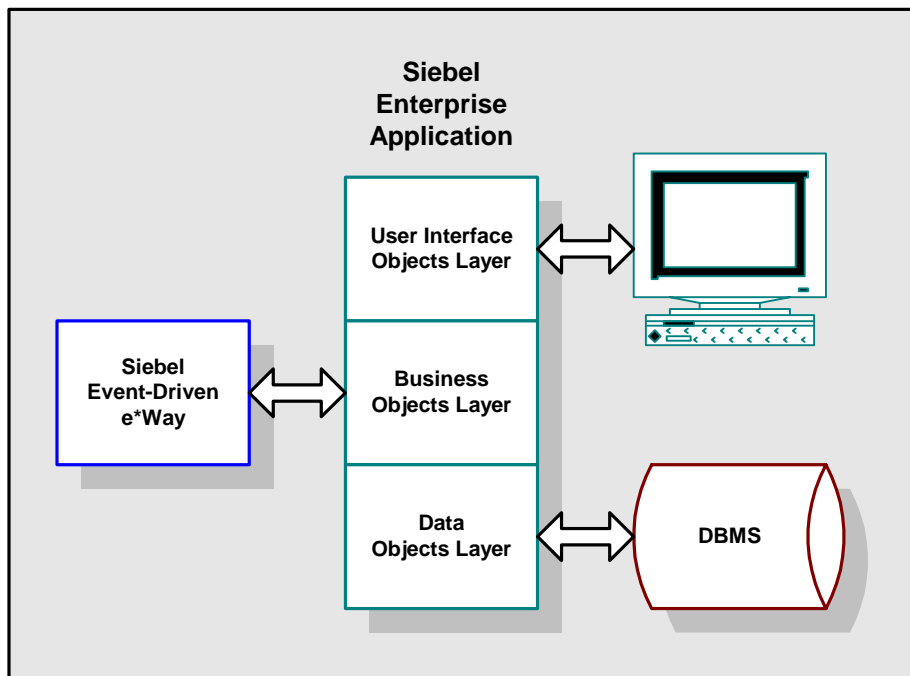
Figure 1   Siebel Object Layers

These three architectural layers, beginning with the lowest layer, are defined as follows:

Data Objects Layer

The Data Objects layer consists of Data Object Definitions, which directly map the data structures from the underlying relational database into Siebel Enterprise Applications, thereby providing access to those structures by Object Definitions in the Business Objects layer. Data Object Definitions insulate both the application and the developer from database administration and restructuring.

Business Objects Layer

The Business Object layer consists of Business Object Definitions, which are built on Data Object Definitions, and selectively combine and associate Data Object Definitions into logical data constructs that are useful for application design. Two of these logical constructs, for example, are Business Components (record structures comprised of Columns from multiple joined Tables) and one-to-many Links between record structures. This is the layer with which theSiebel Event-Driven e*Way interacts.

User Interface Objects Layer

The User Interface Objects layer consists of User Interface Object Definitions, which define the visual interface with which the user interacts.

1.3   e*Way Operation

1.3.1 e*Gate to Siebel

Figure 2 e*Gate-to-Siebel Process Flow
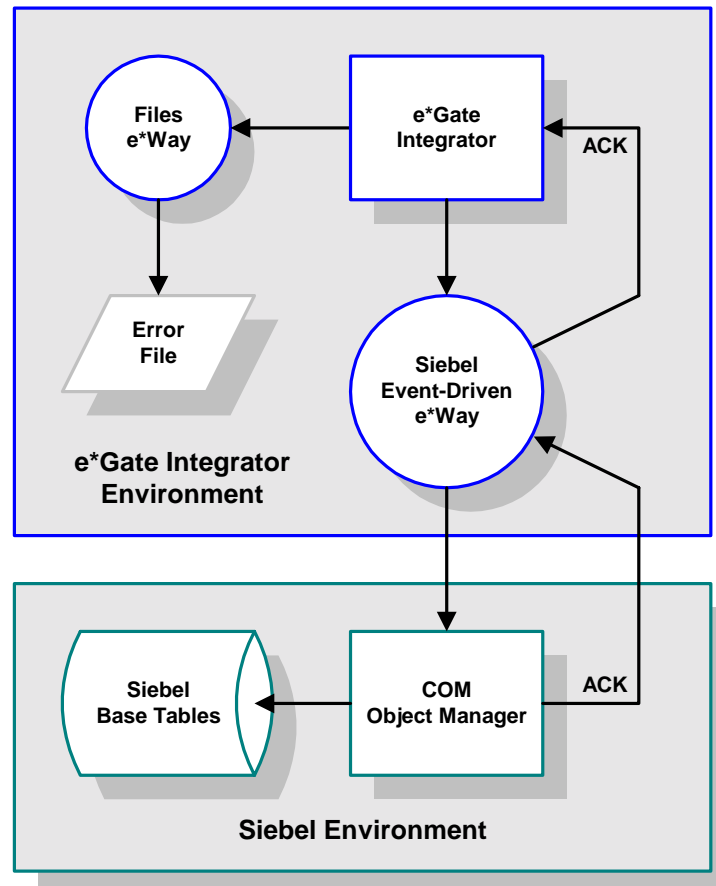
The Siebel Event-Driven e*Way receives data corresponding to a business event in the source application, in the form of message definitions within an ETD. It then invokes the Siebel COM Object Manger to load the business event into the Siebel Base Tables, one message definition at a time.

After the COM Object Manager successfully processes each message definition, a commit to the Siebel database is issued. Once all the message definitions in the message are successfully processed, an **ACK** is sent back to the e*Way, allowing the next message to be submitted for processing.

If a single message definition is not successfully processed, the entire inbound message is failed and be written to a general error file by a File-Handling e*Way. To prevent the message from being resent to the Siebel Event-Driven e*Way, an **ACK** is sent to e*Gate to remove the message from the queue. Also within the failed message, a pointer is saved that identifies the exact message definition that failed.

Figure 3 Error-Correction Process Flow
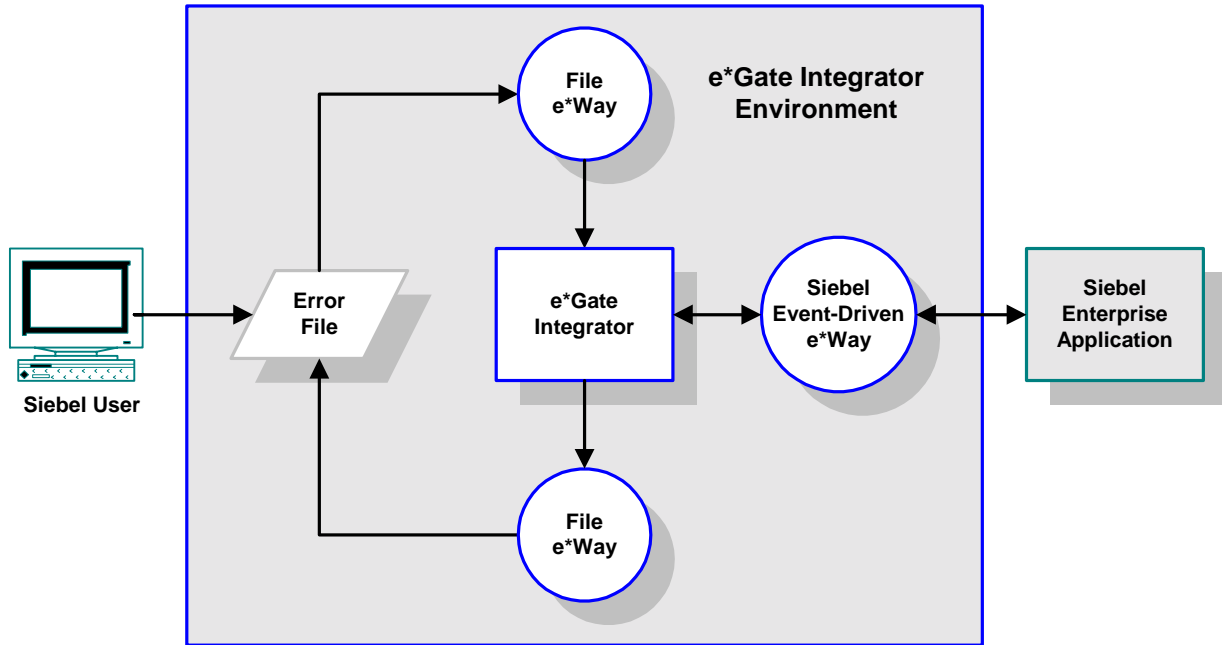
User intervention is required to edit the error file and correct the erroneous data. This can be accomplished using a standard file editor.

1.3.2 **Siebel to e*Gate**

Figure 4 Siebel-to-e*Gate Process Flow
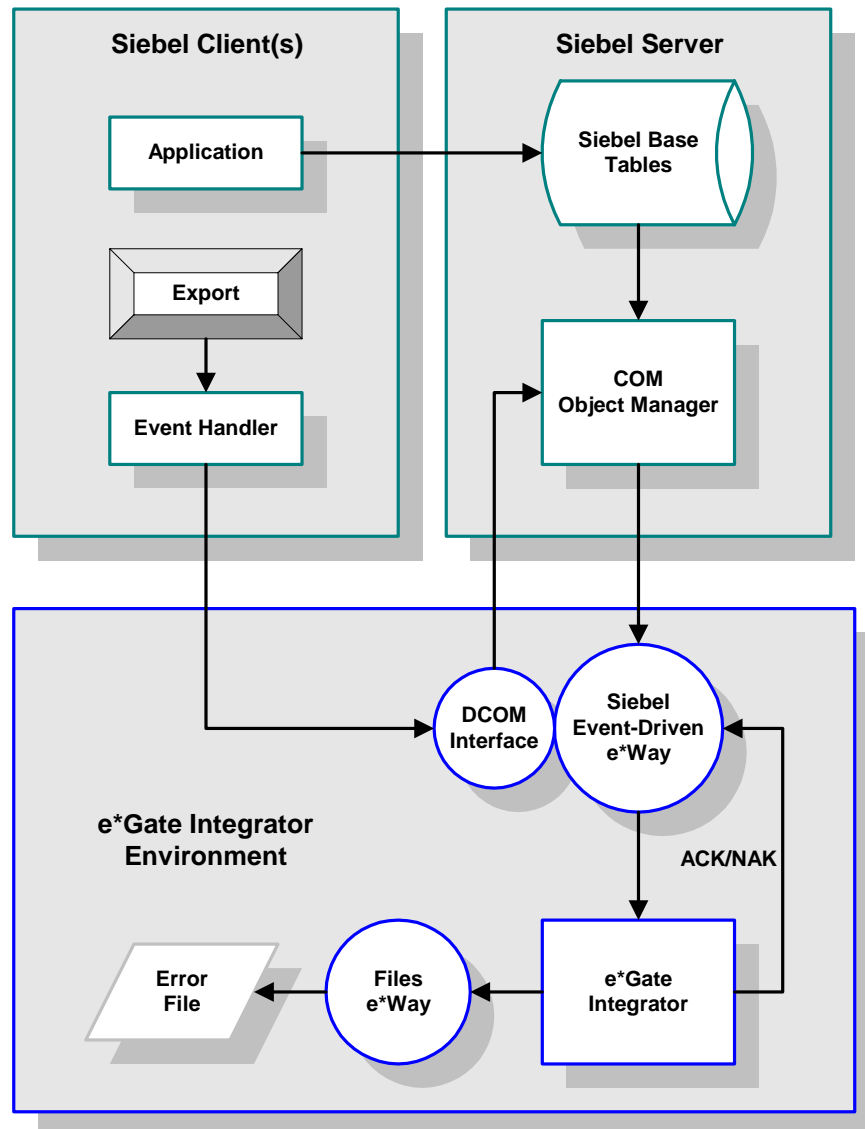
When a business event occurs in Siebel, the transaction is posted to the Siebel Base Tables. The Siebel Event Handler triggers the e*Way's DCOM Interface and passes a message string to it. The message string represents key values for the newly-created record, and is well-defined to encode field name(s), value(s), and logical operator(s)—for example, **Name-LIKE-Account**. (You can add a custom push button to the Siebel Application GUI to initiate this action.)

The message string is passed to a lookup function, which interacts with the Siebel COM interface. The Siebel COM Object Manager then extracts the requested data from the Siebel Base Tables.

The Object Manager then populates the e*Gate ETD for the Siebel business component with the newly-created record. The data is passed to e*Gate for routing to the target e*Way and translation to the target application's ETD. Once an **ACK** is received from e*Gate, the customized status field for the business component is switched to the **Completed** state.

1.4   e*Way Components

The Siebel Event-Driven e*Way is based on SeeBeyond's Generic e*Way Kernel and incorporates the following components:

- The Generic e*Way executable, **stcewgenericmonk.exe** (installed with e*Gate)

- Dynamic load libraries, which provide COM/DCOM access and extend the Generic e*Way Kernel to form the Siebel Event-Driven e*Way

   - **stc_monkcom.dll**

   - **stc_monkdcom.dll**

- An ancillary e*Way executable, **stcewfile.exe** (also installed with e*Gate), for error-file handling—see the *Standard e*Way Intelligent Adapter User's Guide* for information on this e*Way

- The configuration definition file, **stcewsiebeleventdriven.def**

- Monk function scripts and library files, discussed in **Chapter 7**

- Example schema, discussed in **Chapter 3**

For a list of installed files, see **Chapter 2**.

1.5 e*Way Availability

The e*Way Intelligent Adapter for Siebel (Event-Driven) currently supports the following combinations of operating systems and Siebel versions.

English

Table 1 English-language Version

Operating System	Siebel Server		Siebel Front Office		
	5.0	**6.0**	**99.5**	**99.6**	**2000**
Windows 2000 SP1	X	X	X	X	X
Windows 2000 SP2	X	X	X	X	X
Windows NT 4.0 SP6a	X	X	X	X	X

Japanese

Table 2 Japanese-language Version

Operating System	Siebel Server		Siebel Front Office		
	5.0	**6.0**	**99.5**	**99.6**	**2000**
Windows 2000 SP1 (Japanese)	-	X	-	-	X
Windows 2000 SP2 (Japanese)	-	X	-	-	X
Windows NT 4.0 SP6a (Japanese)	-	X	-	-	X

Korean

Table 3 Korean-language Version

Operating System	Siebel Server			Siebel Front Office			
	5.0	**6.0**	**7.0**	**99.5**	**99.6**	**2000**	**7.0**
Windows 2000 SP1 (Korean)	-	X	X	-	-	X	X
Windows 2000 SP2 (Korean)	-	X	X	-	-	X	X
Windows NT 4.0 SP6a (Korean)	-	X	X	-	-	X	X

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\ewsiebelcom directory on the installation CD-ROM for important information regarding this installation.

2.1   System Requirements

To use the e*Way Intelligent Adapter for Siebel (Event-Driven), you need the following:

1   An e*Gate Participating Host, version 4.5.1 or later.

2   A TCP/IP network connection to the Siebel system.

3   Approximately 1 MB of disk space to support e*Way files.

*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, and any external applications performing the processing.*

2.1.1 Environment Configuration

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

2.2 External System Requirements

Note: This e*Way, the Siebel Client, and the database client should all be installed on the same host computer.

2.2.1 Siebel

To use the e*Way Intelligent Adapter for Siebel Event-Driven, you need compatible versions of the following (see also **e*Way Availability** on page 18):

English

- Siebel Server 5.0 or 6.0
- Siebel Front Office 99.5, 99.6, or 2000
- Siebel Enterprise Applications Toolkit

Japanese

- Siebel Server 6.0
- Siebel Front Office 2000
- Siebel Enterprise Applications Toolkit

Korean

- Siebel Server 6.0.2 or 7.0.4
- Siebel Front Office 2000 or 7.0
- Siebel Enterprise Applications Toolkit

Please see the *Siebel System Requirements and Supported Platforms* document for the version of Siebel you are using.

2.2.2 RDBMS

To use the e*Way Intelligent Adapter for Siebel Event-Driven, you need one of the following relational database management systems:

- Oracle
- Microsoft SQL Server

You also need Oracle/SQL Server client software appropriate for the Siebel installation. Please see the *Siebel System Requirements and Supported Platforms* document for the version of Siebel you are using.

2.2.3 External Configuration Requirements

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

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 5). Check **Add-ons**, then click **Next**.

Figure 5 Choose Product Dialog
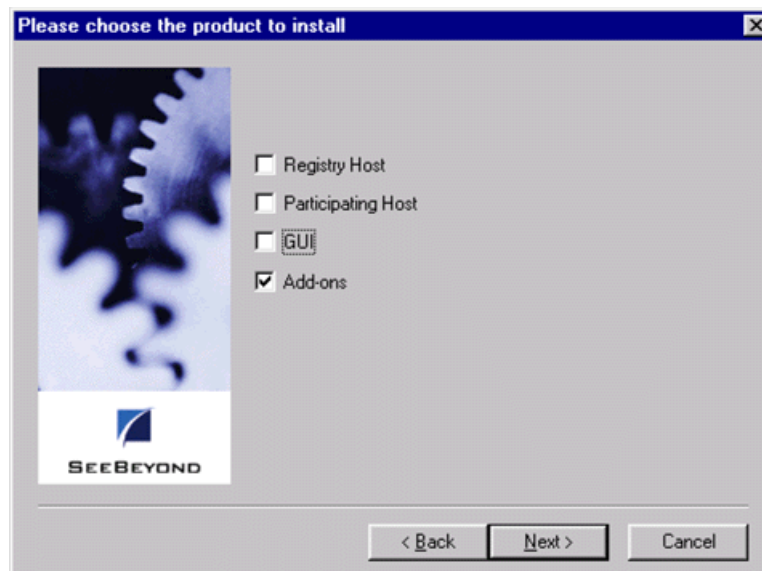
   B If the setup program does not launch automatically, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the following file on the CD-ROM drive (bypassing the **Choose Product** dialog):

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

5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 6). Highlight—*but do not check*—**eWays** and then click **Change**.

Figure 6   Select Components Dialog
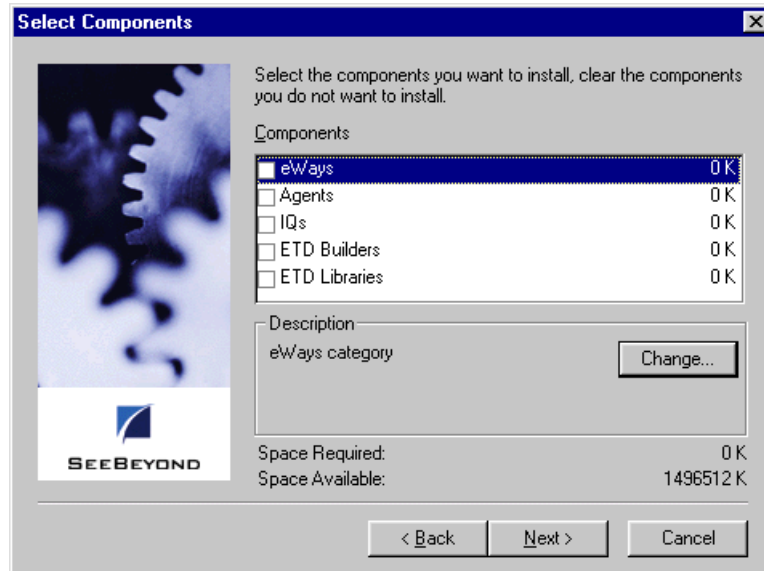
6 When the **Select Sub-components** dialog box appears (see Figure 7), check the **Siebel Event-Driven e*Way**.

Figure 7   Select e*Way Dialog
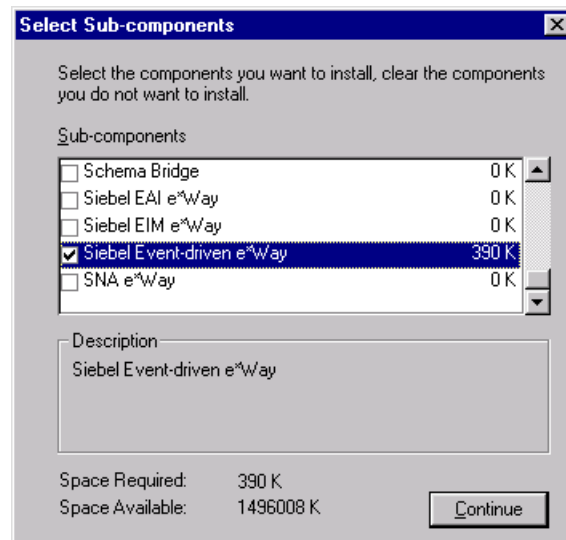
7 Click **Continue**, and the **Select Components** dialog box reappears.

8 Click **Next** and continue with the installation.

2.3.2 **Subdirectories and Files**

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 4   Participating Host & Registry Host

Subdirectories	Files
\bin\	stcsif2ssc.exe stc-monkcom.dll stc-monkdcom.dll
\configs\stcewgenericmonk\	siebelEvent3.6To4.1Rule.txt stcewsiebeleventdriven.def
\monk_library\	ewsiebel.gui
\monk_library\ewsiebeleventdriven\	dcomreg.init sieb-associate.monk sieb-error.monk sieb-get-associate-bc.monk sieb-get-field-values.monk sieb-get-field-values.monk sieb-get-mvg-bc.monk sieb-get-picklist-bc.monk sieb-get-search-expr.monk sieb-query.monk sieb-query2.monk sieb-select-mvg-fields.monk sieb-select-picklist-fields.monk sieb-struct-delete.monk sieb-struct-delete2.monk sieb-struct-get-bc.monk sieb-struct-insert-with-pick.monk sieb-struct-insert.monk sieb-struct-insert2.monk sieb-struct-lookup.monk sieb-struct-lookup2.monk sieb-struct-set-field.monk sieb-struct-set-field2.monk sieb-struct-single-insert.monk sieb-struct-single-insert2.monk sieb-struct-update.monk sieb-struct-update2.monk sieb-struct-write-pick-mvg.monk sieb-struct-write.monk sieb-struct-write2.monk siebel-eventdriven-connect.monk siebel-eventdriven-exchange.monk siebel-eventdriven-init.monk siebel-eventdriven-utils.monk siebel-eventdriven.monk

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

Table 5 Registry Host Only

Subdirectories	Files
\	stcewsiebelcom.ctl

2.4 Registering the DCOM Server

2.4.1 Registering on the e*Gate Host

On the e*Gate installation host, perform following steps once:

1 Locate the registering script in

```
eGate\Server\Registry\repository\default\monk_library\ewsiebeleven
tdriven\dcomreg.init
```

2 Invoke the scripts

```
stctrans dcomreg.init
```

STC_Component.CSTC_DComSvr

This is the interface class for the DCOM server. After a successful registration of DCOM server following entries are made in the Windows registry:

```
HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-006008DFCB04}
        (Default)          "STC DGW COM Server"
        (AppID)            "{02DF8330-3B18-11D3-8DCD-006008DFCB04}"

HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-
006008DFCB04}\LocalServer32
        (Default)          "stcewgenericmonk.exe"

HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-006008DFCB04}\ProgID
        (Default)          "STC_Component.CSTC_DComSvr.1"

HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-
006008DFCB04}\VersionIndependentProgID
        (Default)          "STC_Component.CSTC_DComSvr"

HKEY_CLASSES_ROOT\TypeLib\{02DF8332-3B18-11D3-8DCD-
006008DFCB04}\1.0\0\win32
        (Default)          "E:\eGate\client\bin\stc_monkdcom.dll"

HKEY_CLASSES_ROOT\AppID\{02DF8330-3B18-11D3-8DCD-006008DFCB04}
        (Default)          "STC DGW COM Server"
```

Through the interface, **STCFuncInvoke** is exposed.

```
STCFuncInvoke (Param1, Param2, Param3, Param4)
```

where   Param1: Name of the monk function to invoke (String)

Param2: Message to be passed in the function (String)

Param3: Returned message (String)

Param4: return value (Long)

2.4.2 Registering on the Client Host

1 On the e*Gate distribution CD, locate the setup program at

```
setup\addons\ewsiebelcom\dcom\dcomclientsetup.zip
```

2 Unzip the compressed file

3 Run setup.exe

4 The setup program saves a sample Visual Basic program to the specified directory. You need to supply the Siebel Event-Driven e*Way server host name when asked.

Note: *The invoked Monk function is assumed to be loaded and made available by the e*Way.*

After a successful registration of DCOM server following entries are made in the Windows registry:

```
HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-006008DFCB04}
        (Default)           "STC DGW COM Server"
        (AppID)            "{02DF8330-3B18-11D3-8DCD-006008DFCB04}"

HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-
006008DFCB04}\LocalServer32
        (Default)           "stcewgenericmonk.exe"

HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-006008DFCB04}\ProgID
        (Default)           "STC_Component.CSTC_DComSvr.1"

HKEY_CLASSES_ROOT\CLSID\{02DF8330-3B18-11D3-8DCD-
006008DFCB04}\VersionIndependentProgID
        (Default)           "STC_Component.CSTC_DComSvr"

HKEY_CLASSES_ROOT\AppID\{02DF8330-3B18-11D3-8DCD-006008DFCB04}
        (Default)           "STC DGW COM Server"
        RemoteServerName      "hostname"
```

2.4.3 **Verifying the DCOM Server Registration**

1 To check that the SeeBeyond DCOM Server is registered, in the **Run** dialog type in the command **dcomcnfg** and click **OK**.

Figure 8 Run Dialog
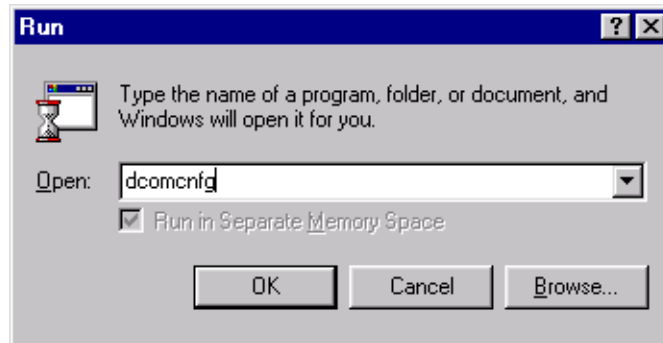
The Distributed COM Configuration Properties dialog lists the registered applications. SeeBeyond (or STC DGW) DCOM Server should be in the list.

Figure 9 Distributed COM Configuration Properties
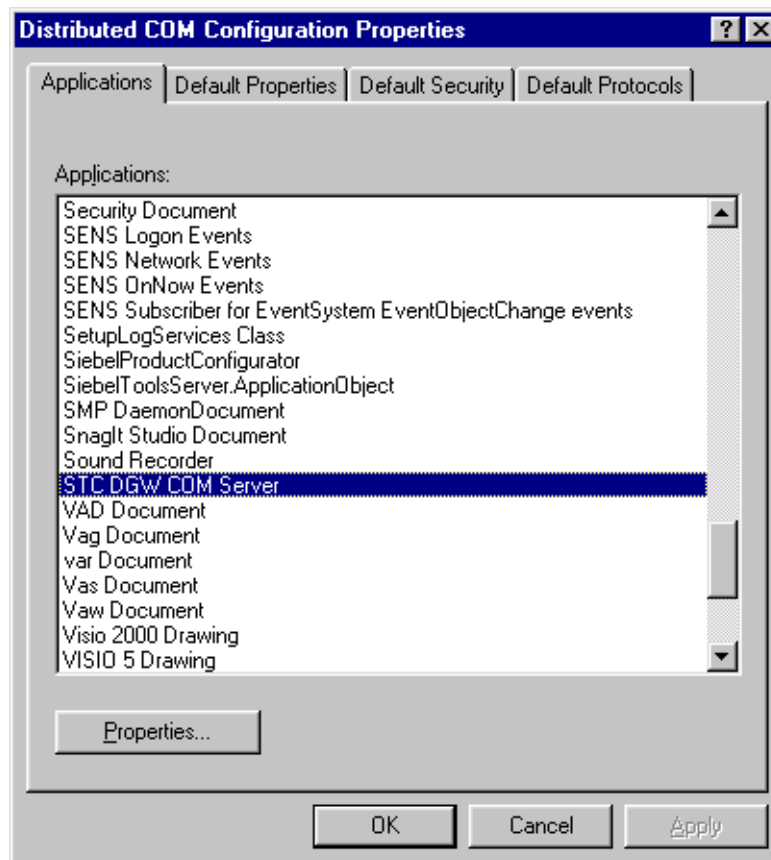
2 Click **Properties**.

3   Select the **Location** tab.

4   Select the **Run application on the following server** check box and type in the name
of the system on which the Siebel Event-Driven e*Way is running.

Figure 10   COM Server Properties - Location tab.
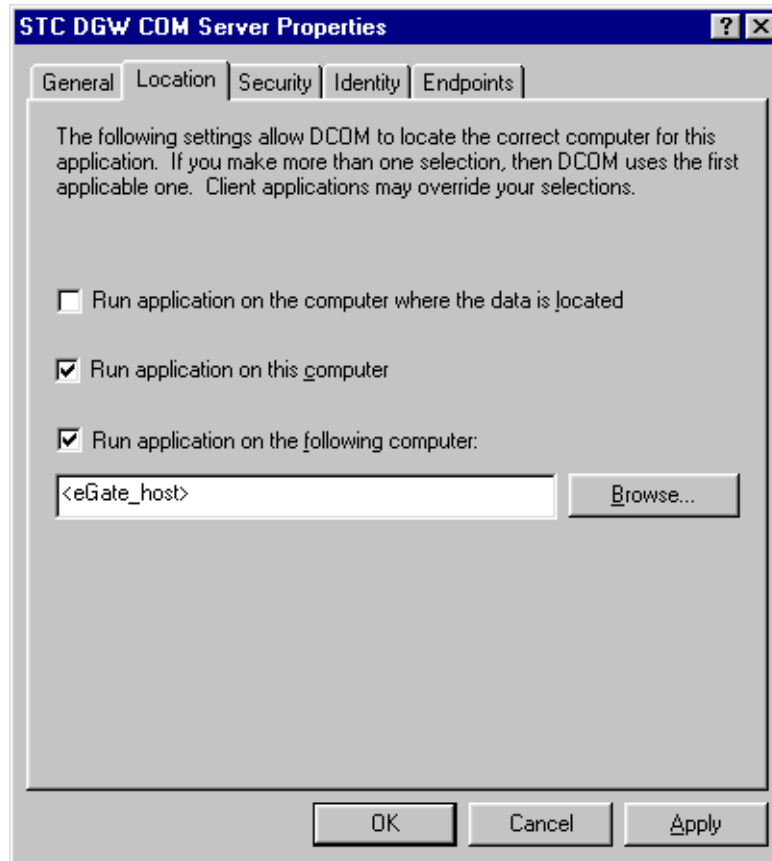
5   Click **OK**.

The DCOM server is now registered on the client computer. Applications on this
computer can now send COM requests to the DCOM server.

2.5 Optional Example Files

The installation CD-ROM contains three sample schemas, located in the **samples\ewsiebelcom** directory. To use a schema, you must load it onto your system using the following procedure. See **Sample Schemas** on page 41 for descriptions of the sample schemas and instructions regarding their use.

The sample implementations are:

- **SiebelComAccountPost** (e*Gate-to-Siebel configuration)
- **SiebelComAccountQueryReply** (Siebel-to-e*Gate configuration)
- **SiebelComAccTriggeredQuery** (Siebel-driven, Siebel-to-e*Gate configuration)

Note: *The Siebel Event-Driven e*Way must be properly installed on your system before you can run the sample schema.*

2.5.1 Installation Procedure

1 Invoke the **Open Schema** dialog box and select **New** (see Figure 11).

Figure 11 Open Schema Dialog
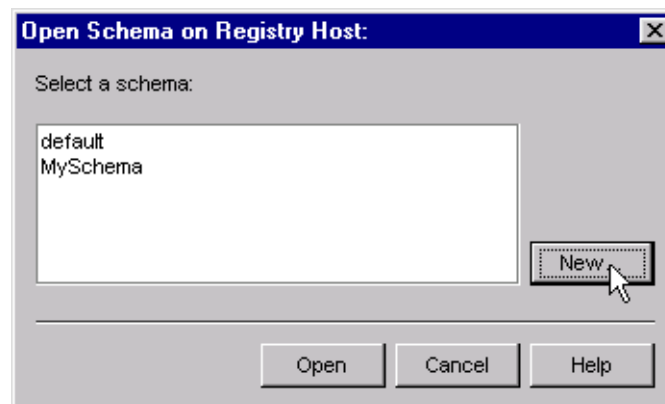
2 Type the name you want to give to the schema (for example, **xxx.Sample**)

3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 12).

Figure 12 New Schema Dialog
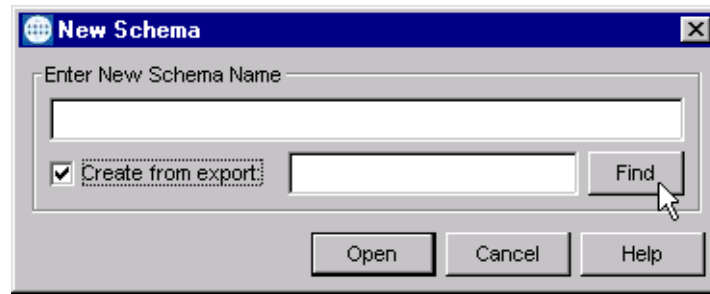
4 Navigate to the desired archive file (***.zip**) and click **Open**.

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.*

2.5.2 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 6 Subdirectories and Files - SiebelComAccountPost

Subdirectories	Files
\	SiebelComAccountPost.ctl
\runtime\configs\stcewfile\	SiebelAccountFeeder.cfg SiebelAccountFeeder.sc SiebelComAccountError.cfg SiebelComAccountError.sc
\runtime\configs\stcewgenericmonk\	SiebelComAccountPost.cfg SiebelComAccountPost.sc
\runtime\monk_libiray\ewsiebeleventdriven\	siebel-eventdriven-account-post.monk
\runtime\monk_scripts\common\	AccountData.ssc AccountError.ssc collab_QToSiebel.tsc sieb-account.ssc

Table 7 Subdirectories and Files - SiebelComAccountQueryReply

Subdirectories	Files
\	SiebelComAccountQueryReply.ctl
\runtime\configs\stcewfile\	SiebelAccountQueryFeederEater.cfg SiebelAccountQueryFeederEater.sc SiebelComQueryErrorEater.cfg SiebelComQueryErrorEater.sc
\runtime\configs\stcewgenericmonk\	SiebelComAccountQueryReply.cfg SiebelComAccountQueryReply.sc
\runtime\monk_libiray\ewsiebeleventdriven\	siebel-com-account-exchange.monk siebel-com-account-query.monk
\runtime\monk_scripts\common\	collab_QReplyToFile.tsc EnqueueSiebelReply.monk sieb-account.ssc SiebelAccountQuery.ssc SiebelAccountReply.ssc SiebelError.ssc

Table 8 Subdirectories and Files - SiebelComAccountTriggeredQuery

Subdirectories	Files
\	SiebelComAccountTriggeredQuery.ctl
\runtime\configs\stcewfile\	SiebelAccountEater.cfg SiebelAccountEater.sc SiebelComErrorEater.cfg SiebelComErrorEater.sc
\runtime\configs\stcewgenericmonk\	SiebelComAccountServer.cfg SiebelComAccountServer.sc
\runtime\monk_libiray\ewsiebeleventdriven\	siebel-com-account-exchange.monk siebel-com-account-query.monk
\runtime\monk_scripts\common\	collab_QReplyToFile.tsc EnqueueSiebelReply.monk sieb-account.ssc SiebelAccountQuery.ssc SiebelAccountReply.ssc SiebelError.ssc

System Implementation

In this chapter we summarize the procedures required for implementing a working system incorporating the Siebel Event-Driven e*Way. Please refer to the *e*Gate Integrator User's Guide*.

3.1 Overview

This e*Way provides a specialized transport component for incorporation in 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.

One or more sample schema, included in the software package, are described at the end of this chapter. These can be used to test your system following installation and, if appropriate, as a template that you can modify to produce your own schema.

This chapter includes the following topics:

Creating a Schema on page 34

Creating Event Types on page 35

Generating Event Type Definitions on page 35

Defining Collaborations on page 39

Creating Intelligent Queues on page 40

Sample Schemas on page 41

3.1.1 Implementation Sequence

Create Schema

Define Event Types

Generate Event Type Definitions

Create & Configure e*Ways

Define & Configure Collaborations

Create Intelligent Queues

Test & Deploy
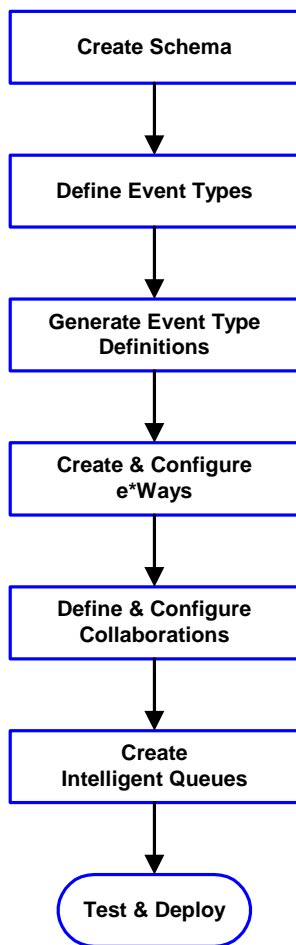
1 The first step is to create a new Schema—the subsequent steps apply only to this Schema (see **Creating a Schema** on page 34).

2 The second step is to define the Event Types you are transporting and processing within the Schema (see **Creating Event Types** on page 35).

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 35).

4 The fourth step is to create and configure the required e*Ways (see **Chapter 4**).

5 Next is to define and configure the Collaborations linking the Event Types from step 2 (see **Defining Collaborations** on page 39).

6 Now you need to create Intelligent Queues to hold published Events (see **Creating Intelligent Queues** on page 40

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 Enterprise Manager 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 Enterprise Manager.

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 (Figure 13) and **Open** an existing schema, or click **New** to create a new schema.

Figure 13   Open Schema Dialog
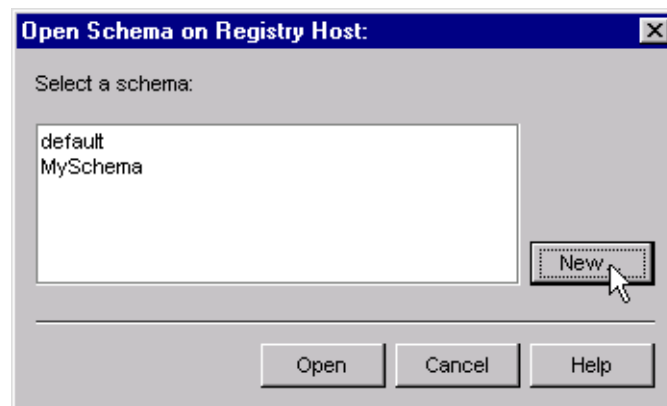
2   Clicking **New** invokes the **New Schema** dialog box (Figure 14).

Figure 14   New Schema Dialog
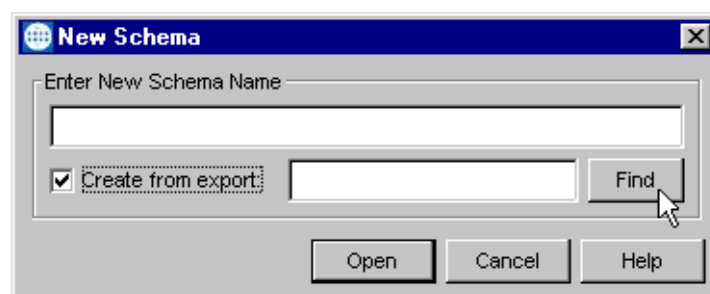
3   Enter a new schema name and click **Open**.

4   The e*Gate Enterprise Manager 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 Enterprise Manager 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 Enterprise Manager'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

For the interface design, the Event structure inbound to Siebel is a superstructure consisting of multiple substructures corresponding to Event Type Definitions. This is necessary because a single Event from the source system may require multiple definitions to load the data into Siebel.

To facilitate the mapping of Siebel message definitions, the Siebel e*Way Event Type Definition Builder is used to build data structures based on Siebel Business Components.

Generating an Event Type Definition is a two-step process:

1   Using Siebel Tools to archive specific business components in a **.sif** file.

2   Using the e*Way ETD Builder to create an ETD from the **.sif** file.

See the *e*Gate Integrator User's Guide* for additional information about Event Type Definitions and the e*Gate ETD Editor.

3.4.1 Using Siebel Tools

Before you can invoke the builder, archive the necessary Business Components that make up one Event Type Definition using Siebel Tools. This manual step creates an ASCII file with a *.sif extension. The *.sif file contains all attributes necessary for the ETD Builder to create a ETD tree within e*Gate.

In Siebel Tools, first select **Business Component**, then **Account**. Select **Repository** from the Menu Bar and, from the resulting pull-down menu, select **Export to Archive File**. For more information, consult the appropriate Siebel documentation.

3.4.2 **Using the ETD Builder**

Place the *.**sif** file in a directory that is convenient to access from e*Gate. Use the following procedure to create an Event Type Definition.

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

To create an Event Type Definition using the Build tool

1 Launch the ETD Editor by clicking [icon] in the e*Gate Enterprise Manager tool bar.

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

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

Figure 15   Build Event Type Definition Dialog
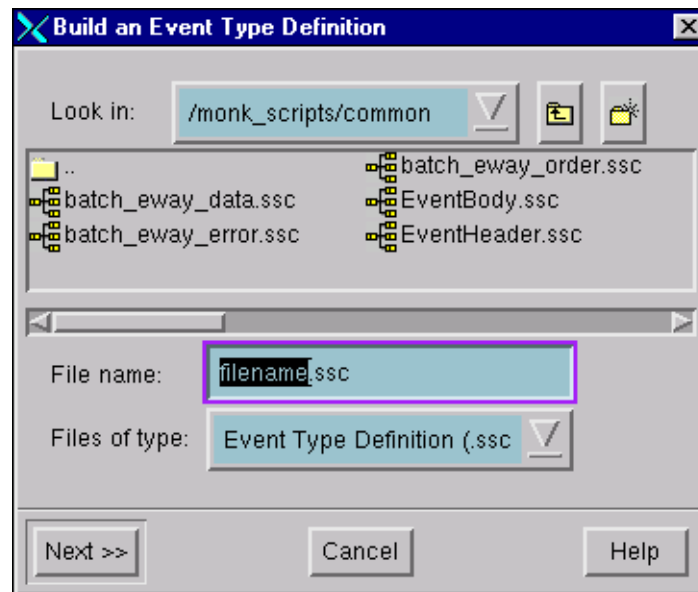
3 In the *File name* box, type the name of the ETD file you want to build.

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

4 Click **Next**. A new dialog box appears, as shown in Figure 16.

Figure 16   Building the ETD
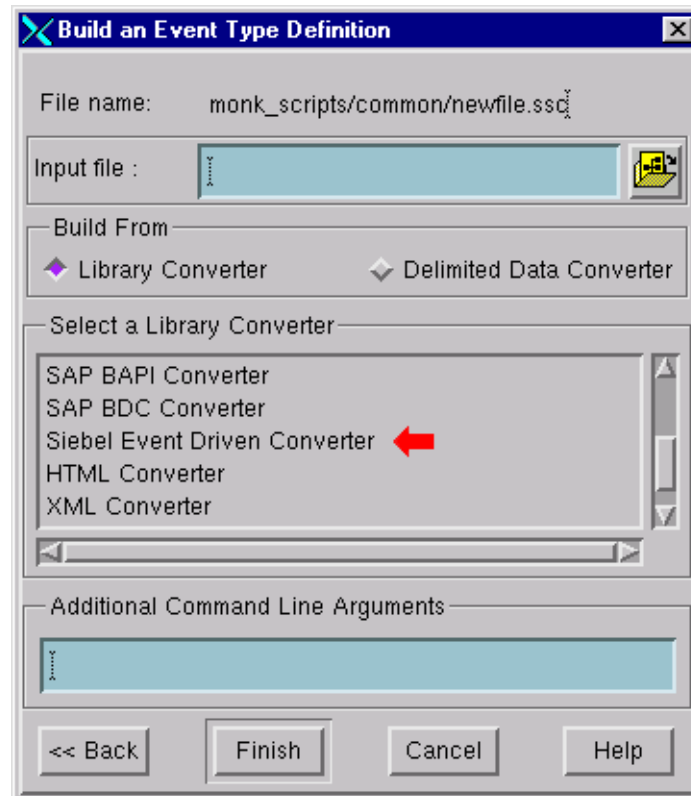
5 Under *Input File*, specify the ***.sif** file to be imported.

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

7 Under *Select a Library Converter*, select **Siebel Event Driven Converter**.

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

9 Click **Finish**.

10 The Siebel Event-Driven Converter Wizard automatically builds the ETD file.

3.4.3 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 Enterprise Manager 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 [icon] in the toolbar).

The Event Type Properties dialog box appears (see Figure 17).

Figure 17   Event Type Properties Dialog Box
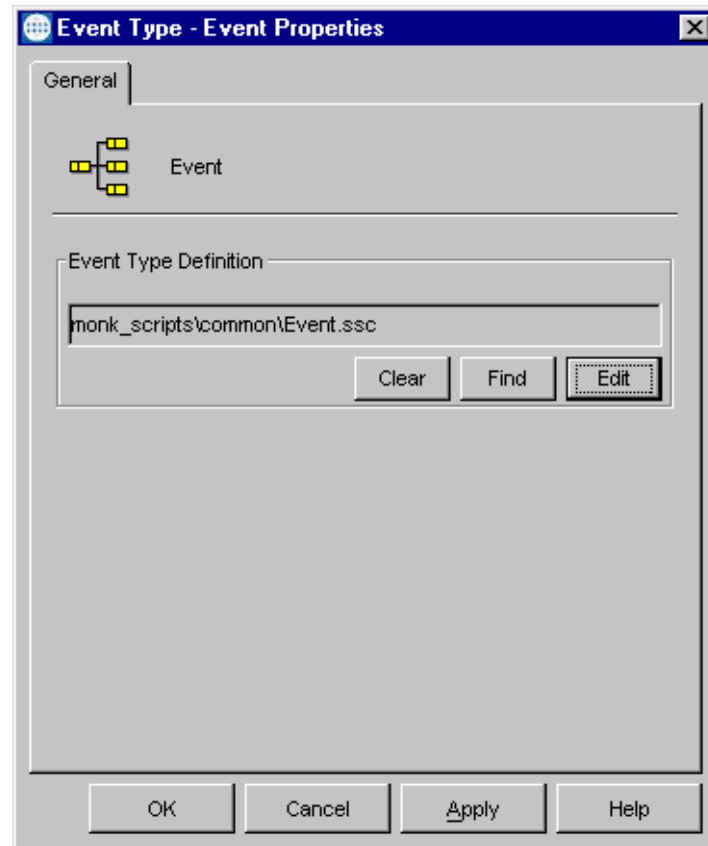
4   Under Event Type Definition, click **Find**, and the Event Type Definition Selection dialog box appears (it is similar to the Windows Open dialog box).

5   Open the **monk_scripts\common** folder, then select the desired file name (***.ssc**).

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 18   Collaborations
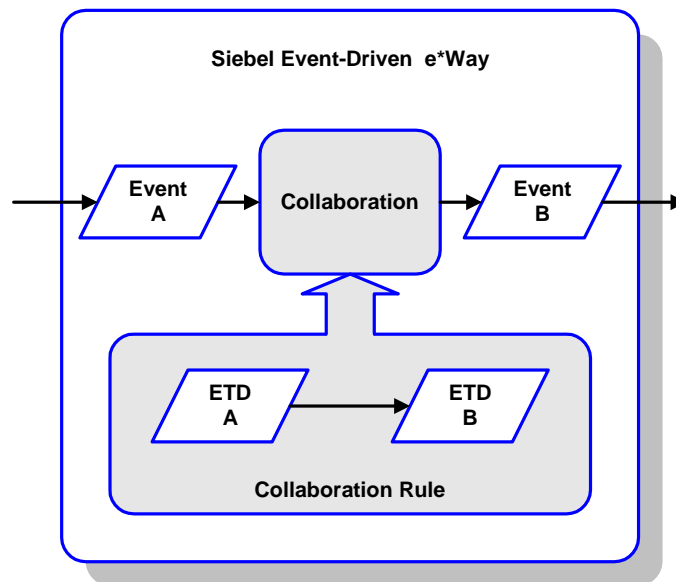
The Collaboration is driven by a Collaboration Rule, which defines the relationship between the incoming and outgoing ETDs. You can use an existing Collaboration Rule, or use the Monk programming language to write a new Collaboration Rule script. Once you have written and successfully tested a script, 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**.

Examples of Collaborations for the Siebel Event-Driven e*Way can be found in **Sample Schemas** on page 41.

3.6 Creating Intelligent Queues

The final step is to create and associate an IQ for the Siebel Event-Driven 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 Schemas

This section refers to sample schemas that are supplied with the Siebel Event-Driven e*Way. These are:

- **SiebelComAccountPost:** e*Gate-to-Siebel example
- **SiebelComAccountQueryReply:** Siebel-to-e*Gate, e*Gate-polling example
- **SiebelComAccTriggeredQuery:** Siebel-to-e*Gate, Siebel-driven example

See **Optional Example Files** on page 29 for information on importing these files.

3.7.1 e*Gate to Siebel Example

In this example, an e*Way reads records describing Siebel *Account* objects from a file and passes them to an Intelligent Queue. A Siebel Event-Driven e*Way receives these records in the form of Events and posts them to the Siebel environment. Any errors are written back to the queue and are picked up by a third e*Way, which writes them to an error file (Figure 19 and **Figure 20 on page 42** illustrate the process).

Figure 19 SiebelComAccountPost Schema
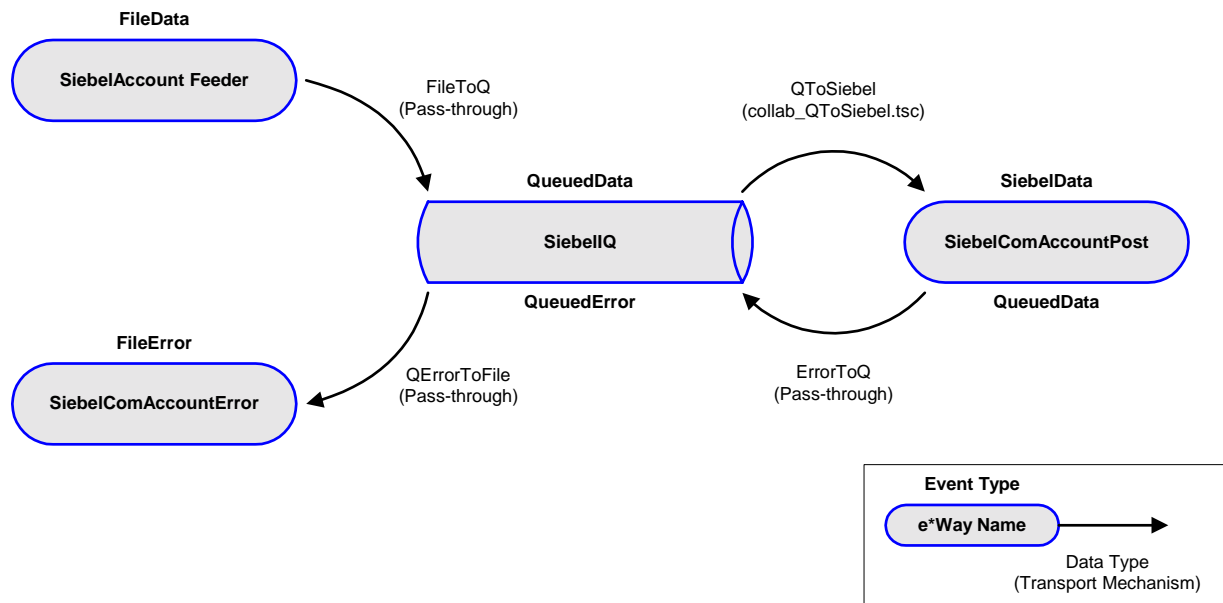
Event Type	Event Structure
FileData	AccountData.ssc
QueuedData	AccountData.ssc
SiebelData	sieb-account.ssc
QueuedError	AccountError.ssc
FileError	AccountError.ssc

Process Flow

Figure 20   e*Gate-to-Siebel Data Flow

```
                    FileData
                       │
                       ▼
              ┌─────────────────┐
              │  SiebelAccount  │
              │     Feeder      │
              └─────────────────┘
                       │
                     Data
                       │
                       ▼
                   ┌───────┐                  ┌──────────────┐         ┌──────────┐
                   │       │      Error       │  SiebelCom   │         │ FileError│
                   │SiebelIQ│ ───────────────▶│ AccountError │ ──────▶ │          │
                   └───────┘                  └──────────────┘         └──────────┘
                    │   ▲
                  Data  Error
                    │   │
                    ▼   │
              ┌─────────────────┐
              │   SiebelCom     │
              │  AccountPost    │
              └─────────────────┘
                       │
                       ▼
         ┌──────────────────────────────────┐
         │                      Siebel       │
         │   ┌──────────┐      ┌────────┐    │
         │   │COM Object│      │ Siebel │    │
         │   │ Manager  │ ───▶ │  Base  │    │
         │   │          │      │ Tables │    │
         │   └──────────┘      └────────┘    │
         └──────────────────────────────────┘
```
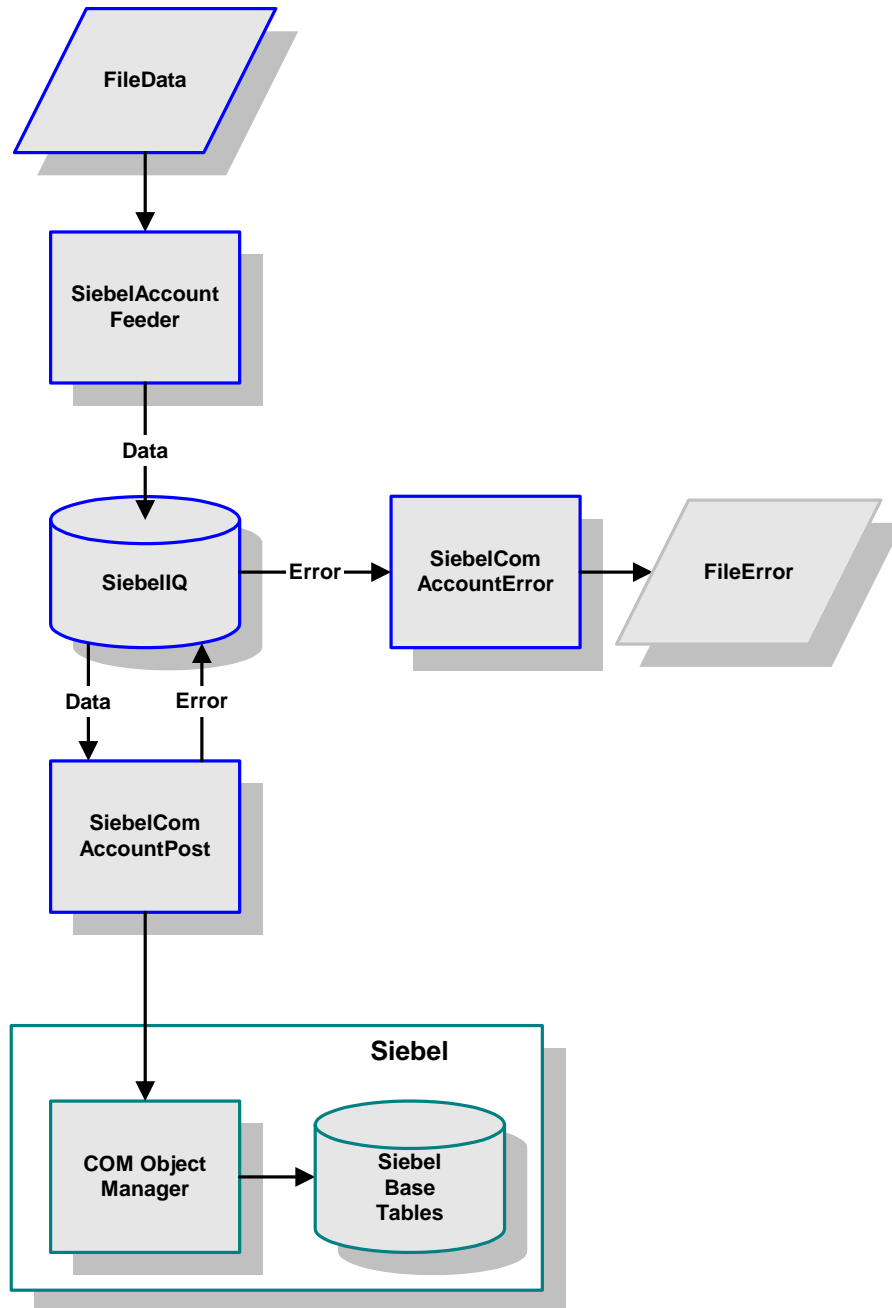
1   The information to be passed to the Siebel environment is stored in a flat file.

Each Event Type Definition in this file contains a repeating node; in this example, accounts information. Therefore, one event can pass data about zero, one, or many Siebel accounts. The ETD **AccountSource.ssc**, as built with the ETD Editor, is shown **Figure 21 on page 43**.

2   A separate e*Way, **SiebelAccountFeeder,** passes the Events (of type **FileData**) to the IQ. The collaboration used performs a byte-by-byte duplication; no processing is done. Therefore, the resulting Event, of type **QueuedData**, is identical to the source Event.

Figure 21   AccountData Event Definition
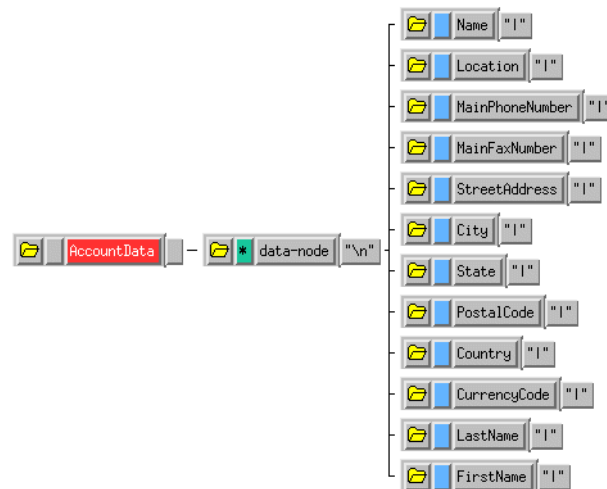
3   The Collaboration **QToSiebel** subscribes to Events of type **QueuedData** in the Intelligent Queue and publishes them to the Siebel Event-Driven e*Way, **SiebelComAccountPost.**

This collaboration maps the incoming Event Definition to a Siebel Event (type **SiebelData**) in which the main node (**~output%sieb-account.PROJECT.Account. BUSINESS_COMPONENT.Account**) may or may not be repeated.

A portion of the Event structure **sieb-account** is shown Figure 22. It is built automatically by the ETD Builder from a sample Siebel object. For more information about the ETD Builder, see **Generating Event Type Definitions** on page 35. The Collaboration Rule is shown **Figure 23 on page 44**.

Figure 22   sieb-account Event Structure
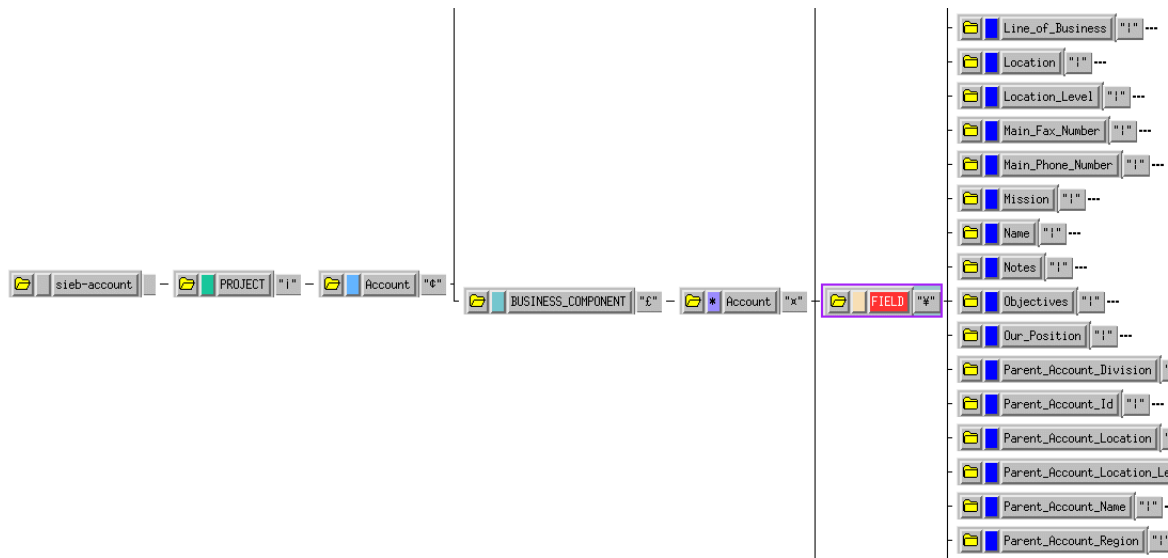
Figure 23   Example Collaboration
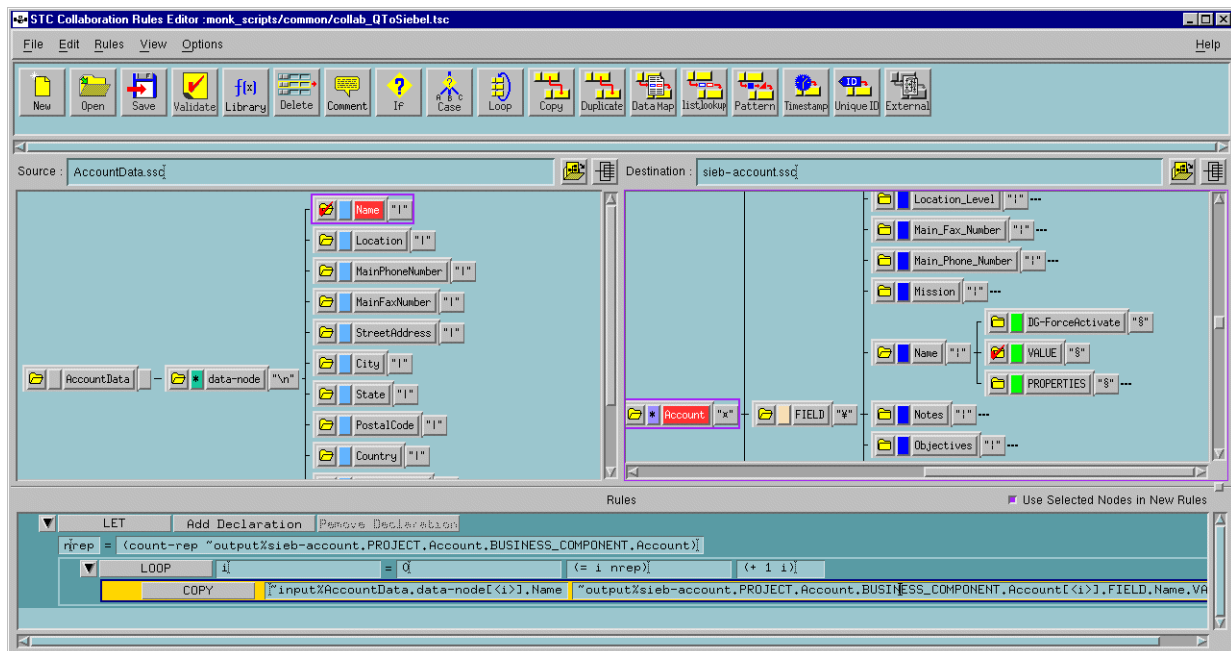
4   The Event is processed via the Monk function, **siebel-eventdriven-account-post,** which is specified as the **Process Outgoing Message Function** in the e*Way Editor. The Monk function **sieb-struct-insert** is called to transfer the data to the Siebel COM Object Manager.

5   If one or more of the **Accounts** objects fails, the remainder of the Event is still processed by Siebel. Additionally, an error Event is produced containing only the failed account object(s). The Collaboration **ErrorToQ** publishes the error Event (type **QueuedError**) to the IQ. A separate File e*Way, **SiebelComAccountError,** subscribes to these Events and writes them to a flat file using the Collaboration **QErrorToFile**.

Collaborations

Four Collaborations are used in this sample schema. The first two, **FileToQ** and **QToSiebel,** propagate valid messages to Siebel. **FileToQ**, is a pass-through service, described in step 2 of the **Process Flow** on page 42. It is diagrammed in Figure 24. The second, **QToSiebel**, is described in step 3 of the **Process Flow** on page 42, and is diagrammed in 25.

Figure 24   FileToQ Collaboration

Source Event	Collaboration	Destination Event
FileData	**FileToQ**	**QueuedData**
AccountData .ssc	**(pass-through)**	**AccountData .ssc**
Source ETD	**Script**	**Destination ETD**
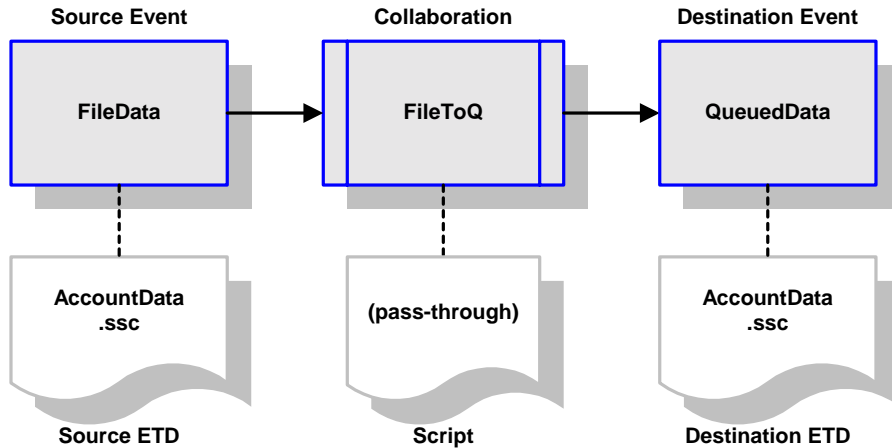
Figure 25   QToSiebel Collaboration

Source Event	Collaboration	Destination Event
QueuedData	**QToSiebel**	**SiebelData**
AccountData .ssc	**collab_QToSiebel .tsc**	**sieb-account .ssc**
Source ETD	**Script**	**Destination ETD**
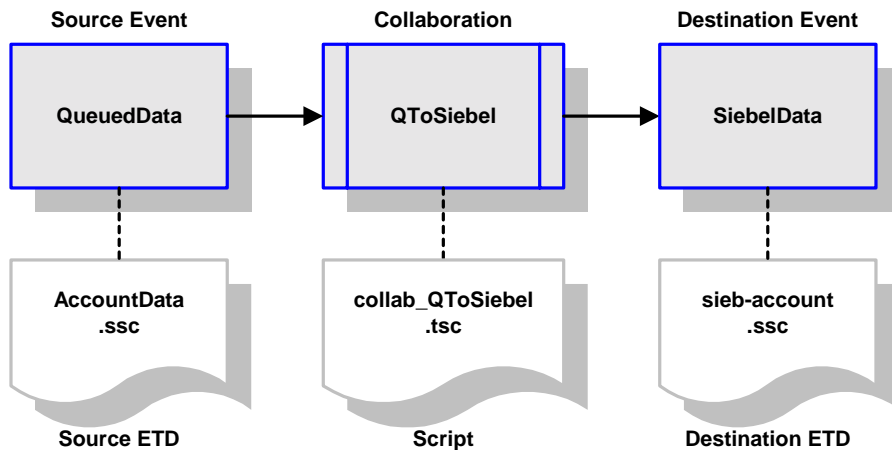
The second two, **ErrorToQ** and **QErrorToFile,** deal with error handling. Both are pass-through services, and are mentioned to in step 5 of the **Process Flow** on page 42. These Collaborations are diagrammed in Figure 25 and Figure 26, respectively.

Figure 26 ErrorToQ Collaboration

Source Event	Collaboration	Destination Event
QueuedData	ErrorToQ	QueuedError
AccountData .ssc	(pass-through)	AccountError .ssc
Source ETD	**Script**	**Destination ETD**
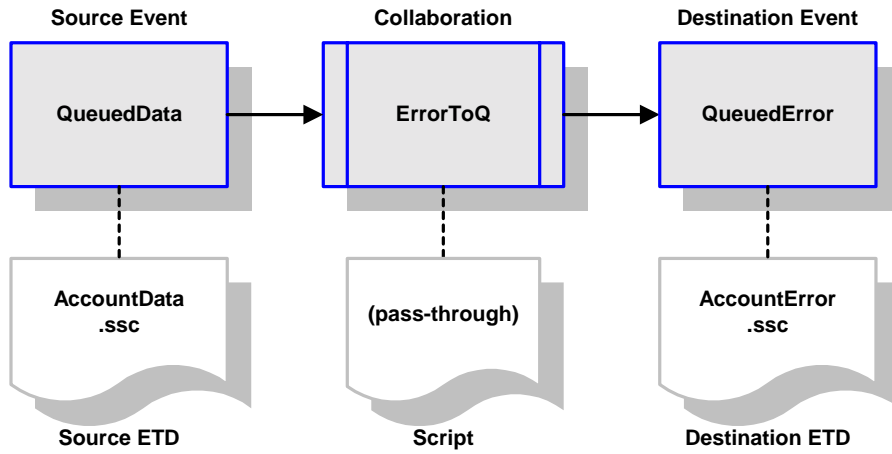
Figure 27 QErrorToFile Collaboration

Source Event	Collaboration	Destination Event
QueuedError	QErrorToFile	FileError
SiebelError .ssc	(pass-through)	SiebelError .ssc
Source ETD	**Script**	**Destination ETD**
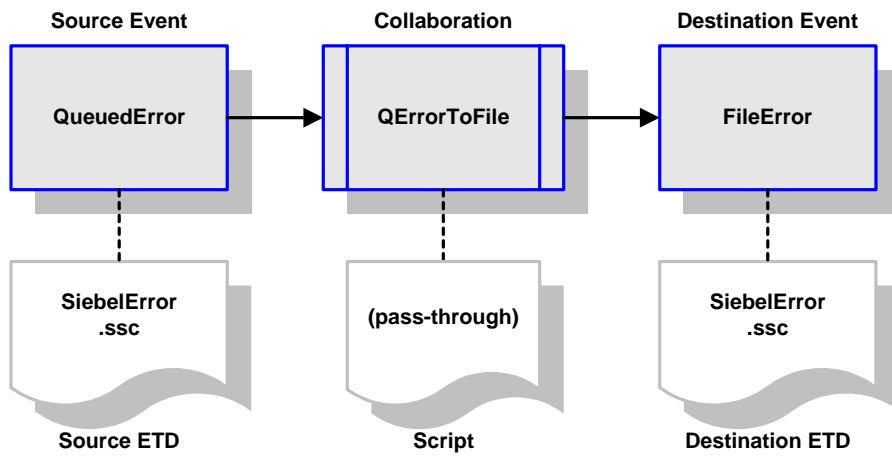
3.7.2 Siebel to e*Gate: Query-Reply Example

In this example, one or more queries are read from a file, **query.fin**, and the reply returned to another file, **reply%d.dat**. (In a real-life environment, an application would send the query and the result would be returned to it by e*Gate.) Figure 28 and Figure 29 illustrate the process.

Figure 28   SiebelComAccountQueryReply Schema
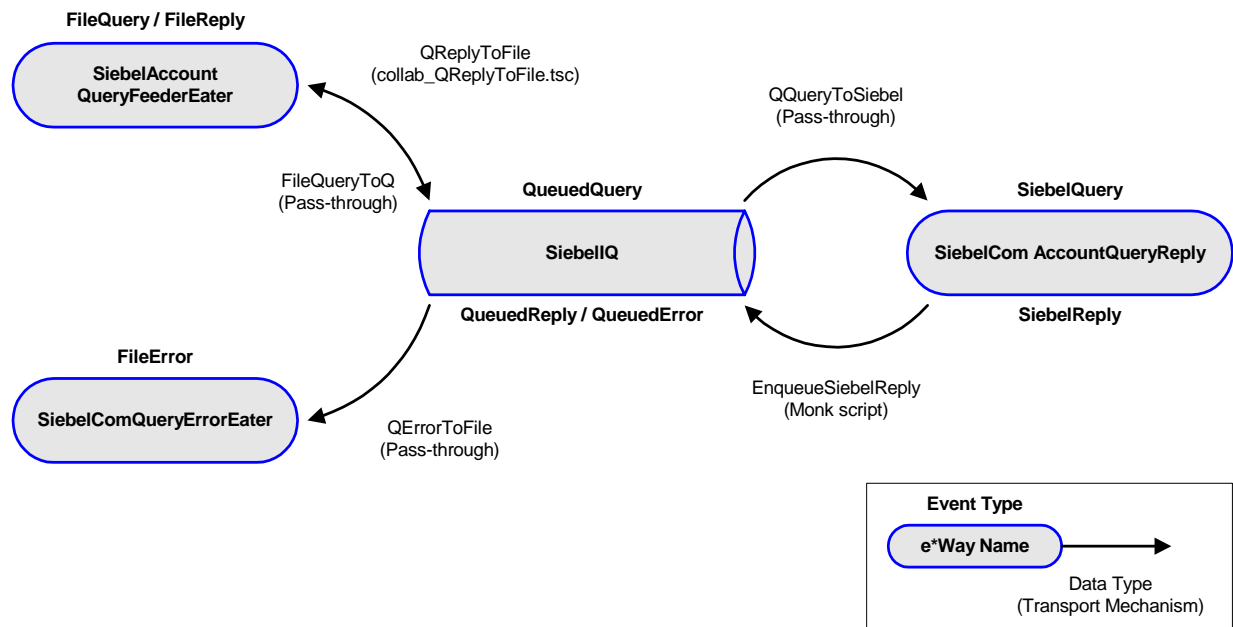
Event Type	Event Structure
FileQuery	SiebelAccountQuery.ssc
QueuedQuery	SiebelAccountQuery.ssc
SiebelQuery	SiebelAccountQuery.ssc
SiebelReply	sieb-account.ssc
QueuedReply	sieb-account.ssc
FileReply	SiebelAccountReply.ssc
QueuedError	SiebelError.ssc
FileError	SiebelError.ssc

Process Flow

Figure 29   Siebel-to-e*Gate Query-Reply Example
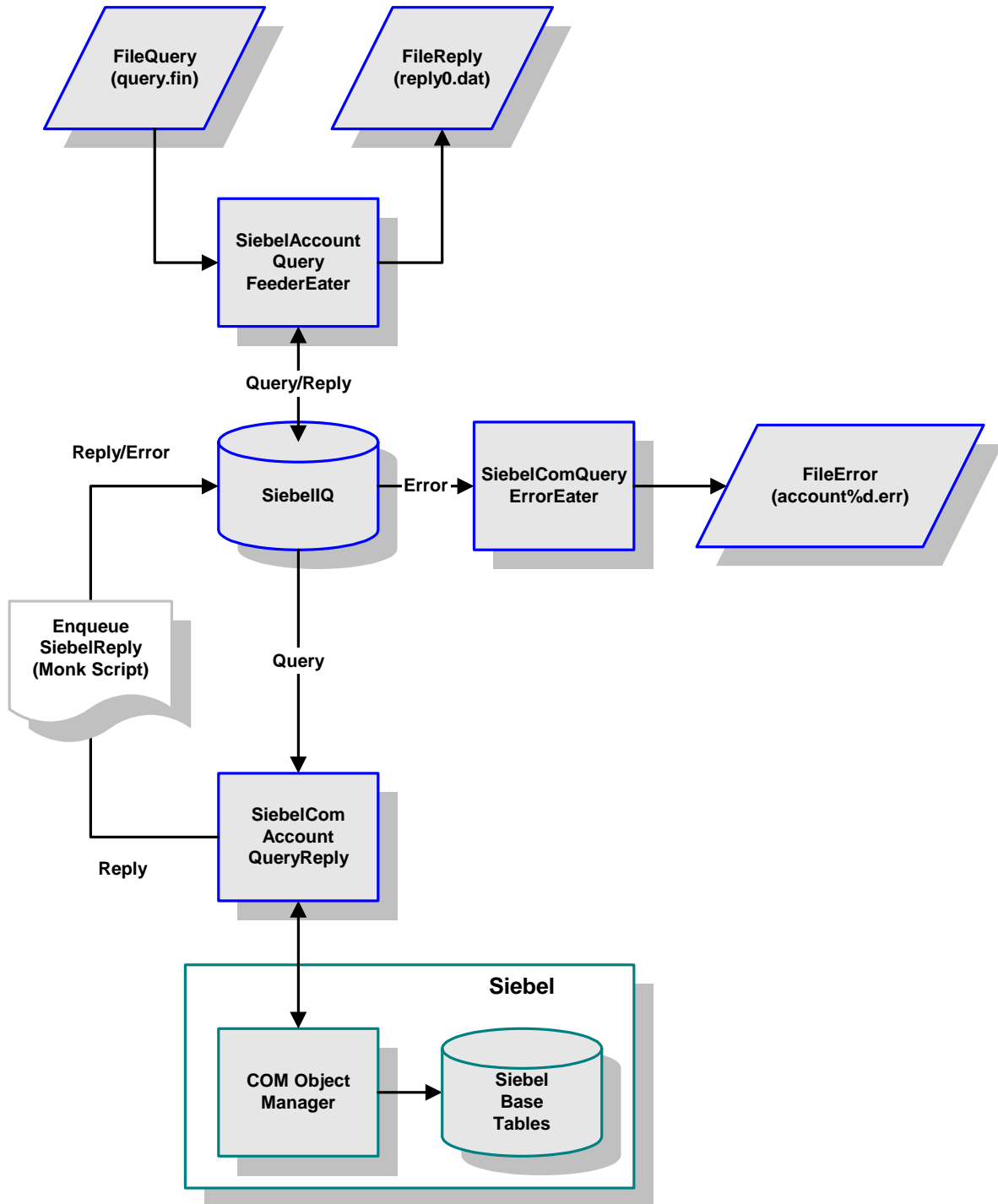
1 Queries to be passed to the Siebel environment are stored in the flat file, **query.fin**, which can contain one or more queries.

2 Queries are passed, one at a time, to the IQ by a bidirectional File e*Way, **SiebelAccountQueryFeederEater**. The collaboration used, **FileQueryToQ**, performs a byte-by-byte duplication; no processing is done to generate **QueuedQuery** Events.

3 The Siebel Event-Driven e*Way, **SiebelComQueryReply**, subscribes to the **QueuedQuery** Event Type in the IQ. Again, this Event is passed unprocessed into the e*Way as **SiebelQuery** Events.

4 On receipt of a query Event, the e*Way calls a Monk function, **siebel-com-account-query**, with the query as the argument. The query is then stored in a global Monk variable (**siebel-current-query**) and the scheduler is invoked. In turn, this calls the Monk function **siebel-com-account-exchange**, which first creates a **sieb-account** structure internally.

   In this example, we are only interested in a small selection of the fields within a Siebel account object, and they are tagged. For example:

```
(insert "Y"
     ~sieb-account-msg%sieb-
account.PROJECT.Account.BUSINESS_COMPONENT.Account[0].FIELD.Main_P
hone_Number.DG-ForceActivate "")
```

   indicates that we require the **Main_Phone_Number** field.

5 The query is passed to the Siebel COM Object Manager. The collaboration used, **QQueryToSiebel**, performs a byte-by-byte duplication; no processing is performed on the **QueuedQuery** Events.

6 A string is returned from the Monk function:

   ▪ If no error occurs, the reply from **sieb-struct-lookup** is mapped to a string and returned.

   ▪ If an error is detected, a message string is constructed with the first 11 characters being **SIEBELERROR**. This string is then returned from the Monk function.

7 The Monk Collaboration **EnqueueSiebelReply** tests the first 11 characters of the message to see if they match **SIEBELERROR**:

   ▪ The the message starts with **SIEBELERROR**, the Monk function **iq-put** is called, and the error is placed on the IQ as an event of type **QueuedError.**

   ▪ Valid replies are published to the IQ as the default Event Type for this collaboration, **QueuedReply**.

8 The queued Events are then handled as follows:

   ▪ **QueuedReply** Events are picked up by the bidirectional File e*Way that initiated the query, transformed into an acceptable format, and written to a flat file.

   ▪ **QueuedError** Events are subscribed to by another File e*Way, which simply takes the error Events from the IQ and writes them to a flat file.

Collaborations

Two pass-through Collaborations are used in the query part of this sample schema: **FileQueryToQ** and **QQueryToSiebel**. These are described in steps 2 and 5 of the **Process Flow** on page 48, and are diagrammed below in Figure 30 and Figure 31, respectively.

Figure 30   FileQueryToQ Collaboration
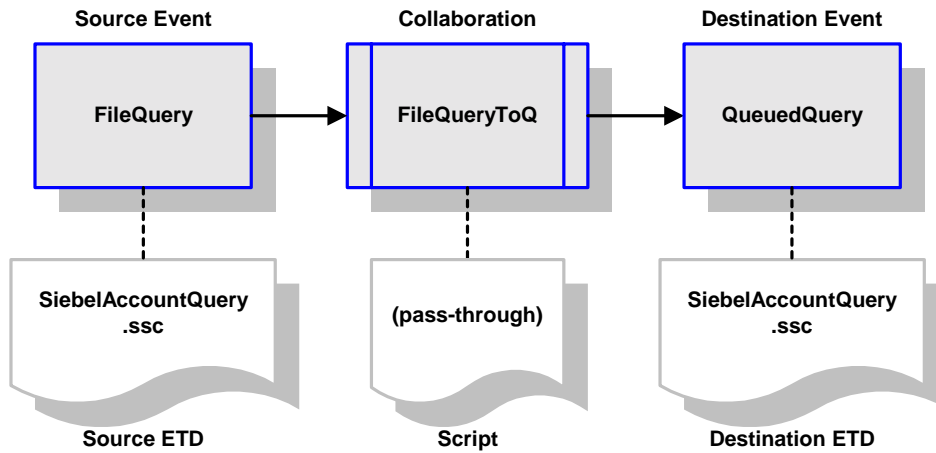
Figure 31   QQueryToSiebel Collaboration
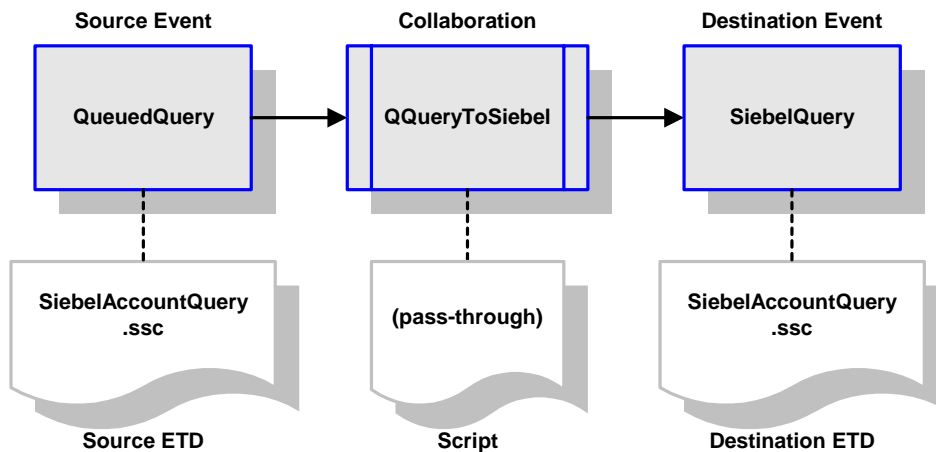
Three Collaborations are used in the data extraction part of this sample schema. The first, **EnqueueSiebelReply**, is a "non-standard" Collaboration in that it contains a conditional clause, and is defined by the Monk script **EnqueueSiebelReply.monk**. Depending upon the outcome of the condition, it is followed by either of two subsequent Collaborations, **QReplyToFile**, or **QErrorToFile**. These are described in context in steps 7 and 8 of the **Process Flow** on page 48, and are diagrammed in **Figure 32 on page 51**, **Figure 33 on page 51**, and **Figure 34 on page 52**, respectively.

Figure 32 EnqueueSiebelReply Collaboration
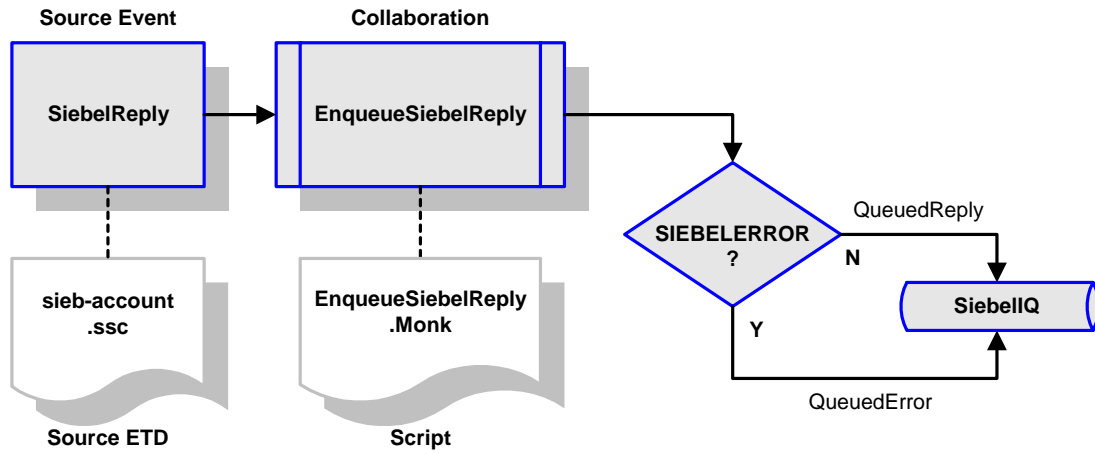
Figure 33 QReplyToFile Collaboration
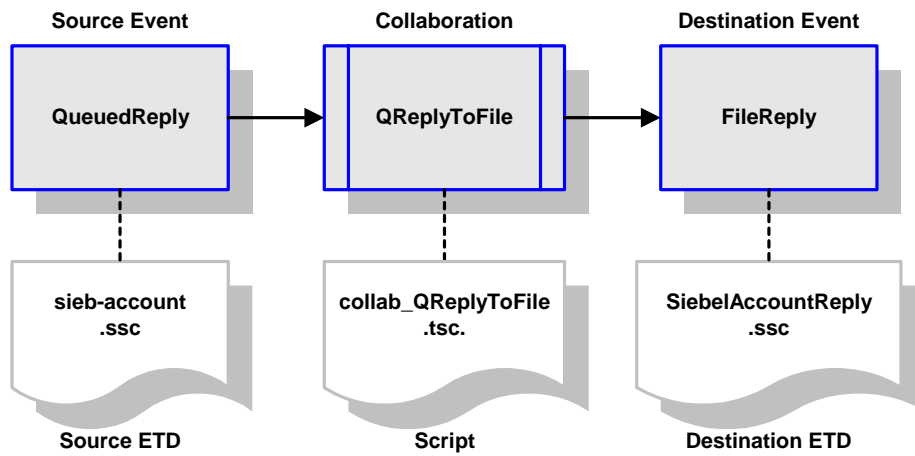
Figure 34   QErrorToFile Collaboration

Source Event	Collaboration	Destination Event
QueuedError	QErrorToFile	FileError
SiebelError .ssc	(pass-through)	SiebelError .ssc
Source ETD	**Script**	**Destination ETD**
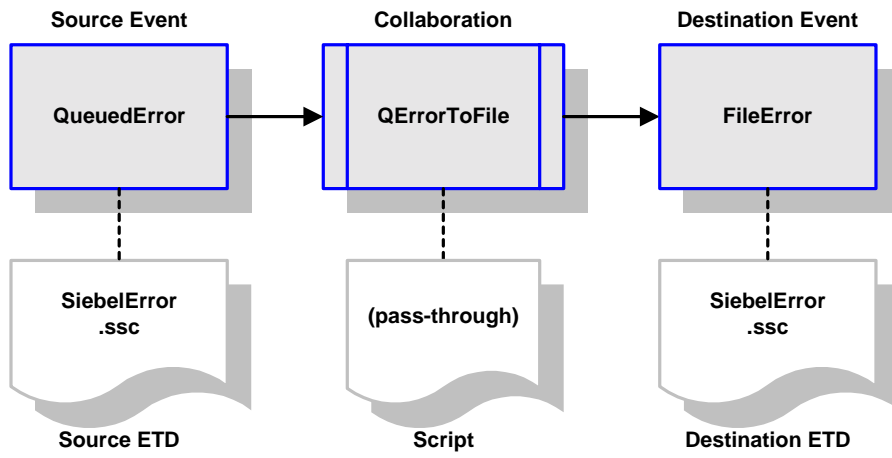
3.7.3 Siebel to e*Gate: COM Server Example

This example shares much of the functionality of the Query-Reply example (see **Siebel to e*Gate: Query-Reply Example** on page 47). The main difference is that the source of the query is not an e*Way, but Siebel VB code (or some other DCOM-compliant application). See **Registering the DCOM Server** on page 25 for more details on DCOM server setup and client setup. Figure 35 and **Figure 36 on page 54** illustrate the process.

Figure 35   SiebelComAccTriggeredQuery Schema
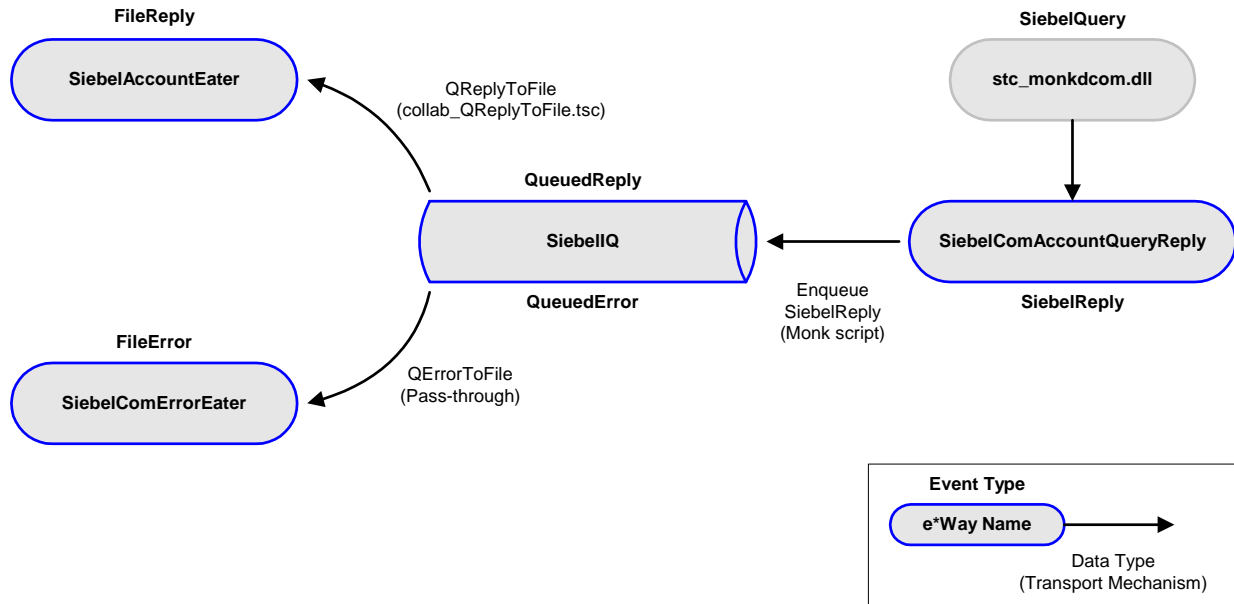
Event Type	Event Structure
FileQuery	SiebelAccountQuery.ssc
QueuedQuery	SiebelAccountQuery.ssc
SiebelQuery	SiebelAccountQuery.ssc
SiebelReply	sieb-account.ssc
QueuedReply	sieb-account.ssc
FileReply	SiebelAccountReply.ssc
QueuedError	SiebelError.ssc
FileError	SiebelError.ssc

Process Flow

Figure 36   Siebel-to-e*Gate COM Server Example
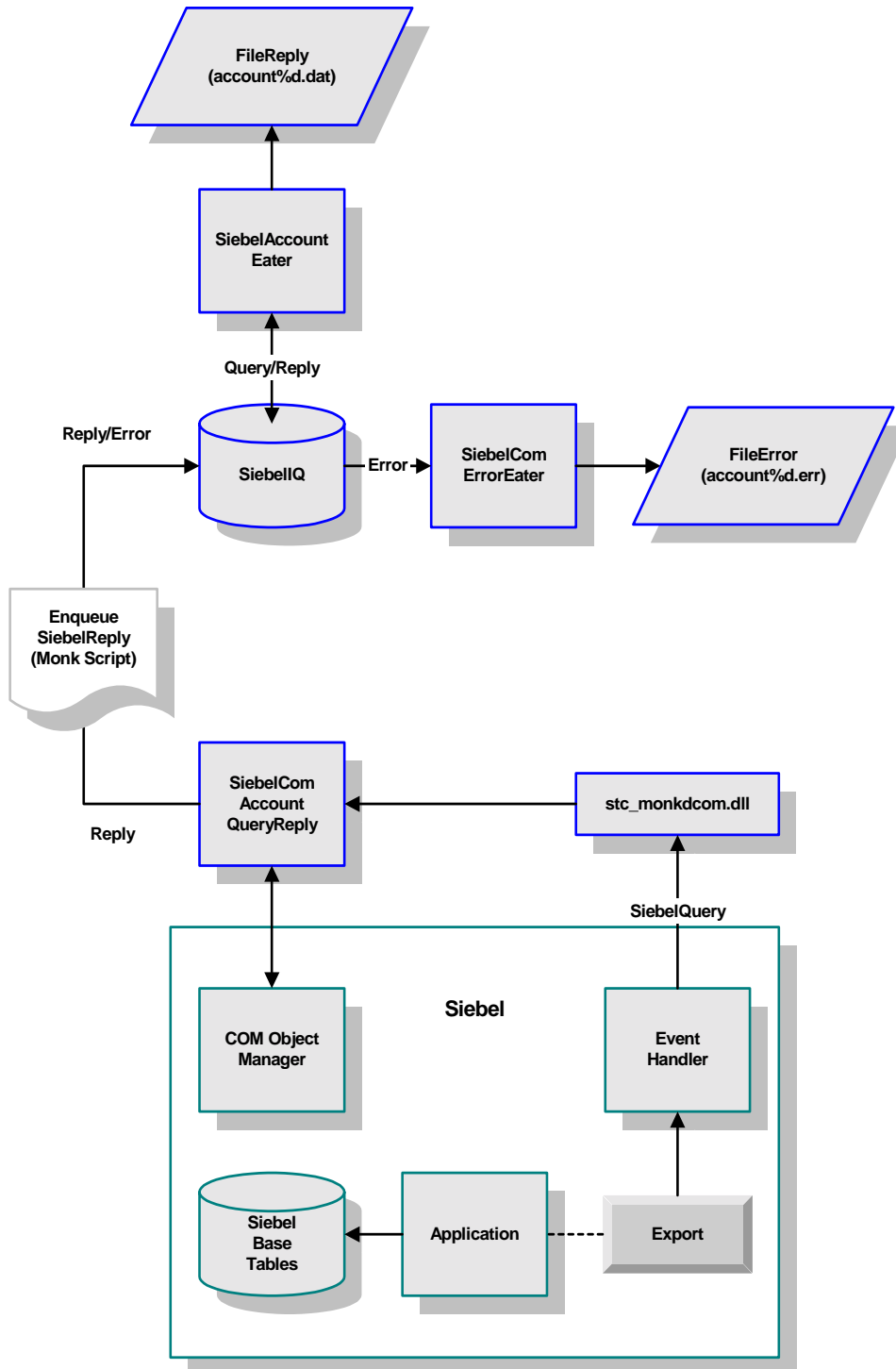
1   On startup, the Siebel Event-Driven e*Way loads the Monk extension DLL containing the COM/DCOM server code, and creates a DCOM server instance.

2   A Siebel application sends a COM request to the server embedded in the e*Way. Among the parameters for this request is the name of a Monk function to invoke, and a string to be passed to it. The example relies on the Monk function being **siebel-com-account-query**, and the query string is the argument.

3   Processing then continues as it would for the Query-Reply example (see **Siebel to e*Gate: Query-Reply Example** on page 47.

Collaborations

In this sample schema, only the data-extraction Collaborations are used; see **Figure 32 on page 51**, **Figure 33 on page 51**, and **Figure 34 on page 52**.

Setup Procedures

This chapter describes the procedures for customizing the e*Way Intelligent Adapter for Siebel (Event-Driven) to operate with your Siebel system.

4.1 Overview

After creating a schema, you must instantiate and configure the Siebel Event-Driven 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 57

Modifying e*Way Properties on page 58

Configuring the e*Way on page 59

Changing the User Name on page 63

Setting Startup Options or Schedules on page 63

Activating or Modifying Logging Options on page 65

Activating or Modifying Monitoring Thresholds on page 66

Troubleshooting the e*Way

Configuration Problems on page 67

System-related Problems on page 68

Monk Errors on page 68

4.2 Setting Up the e*Way

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 Enterprise Manager.

To create an e*Way

1 Open the schema in which the e*Way is to operate.

2 Select the e*Gate Enterprise Manager 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 37 e*Gate Enterprise Manager Window (Components View)
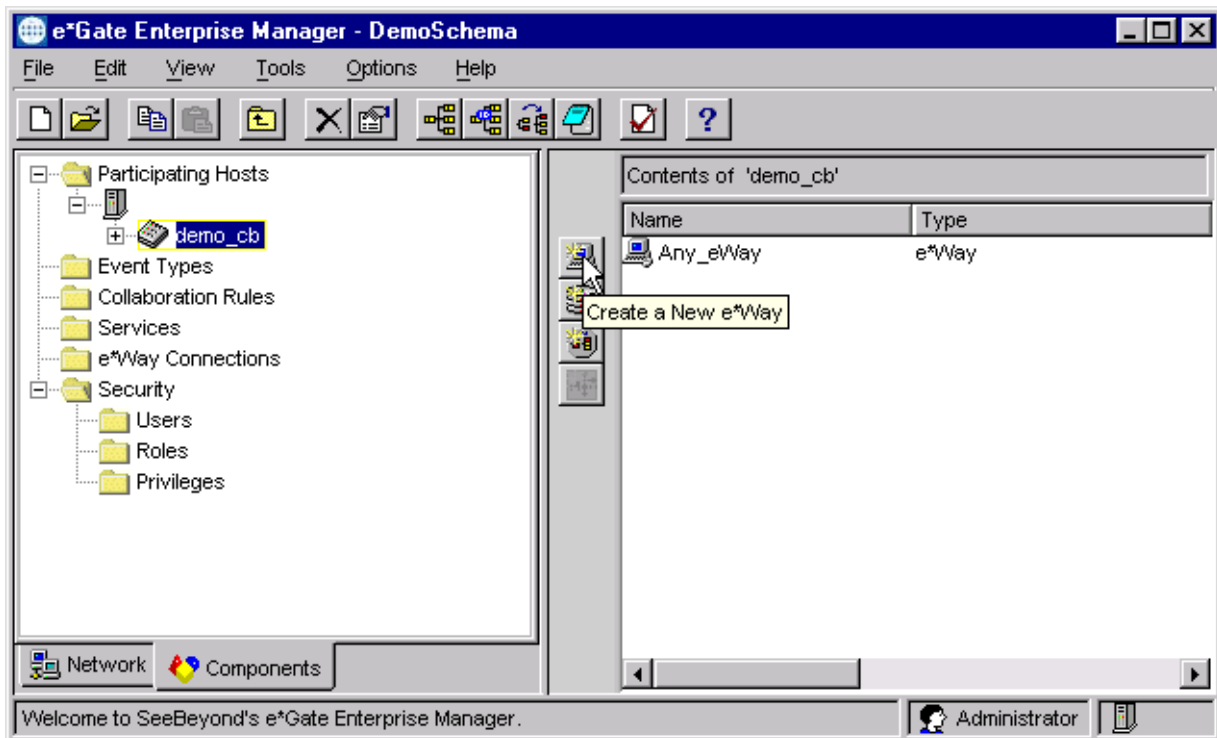
5 On the Palette, click **Create a New e*Way.**

6 Enter the name of the new e*Way, then click **OK**.

7 All further actions are performed in the e*Gate Enterprise Manager 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 38).

Note: *The executable and default configuration files used by this e*Way are listed in*
          e*Way Components *on page 17.*

Figure 38   e*Way Properties (General Tab)
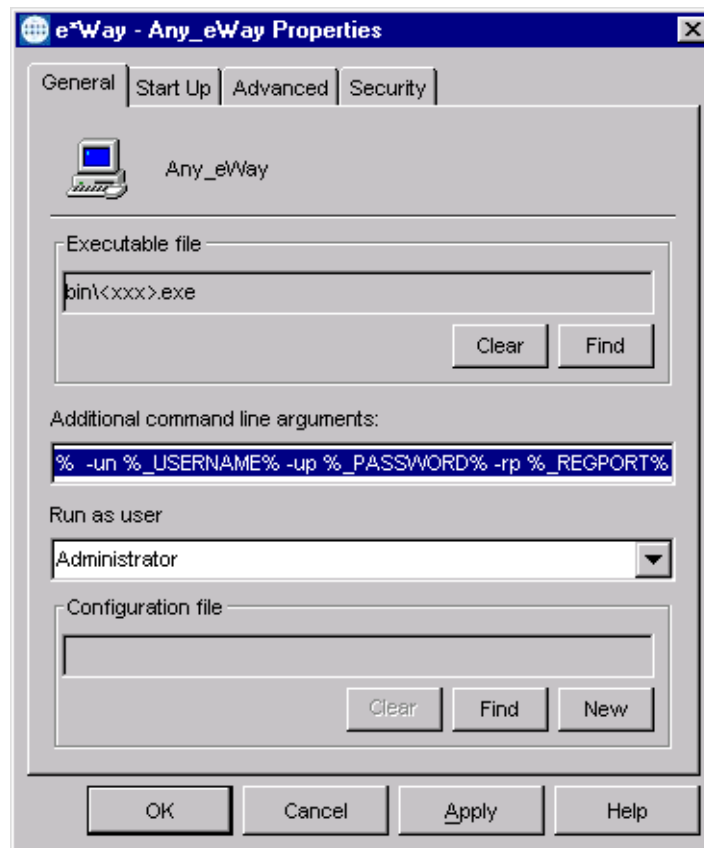
2   Make the desired modifications, then click **OK**.

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

Note: *The executable and default configuration files used by this e*Way are listed in* **e*Way Components** *on page 17.*

Figure 39   e*Way Properties - General Tab
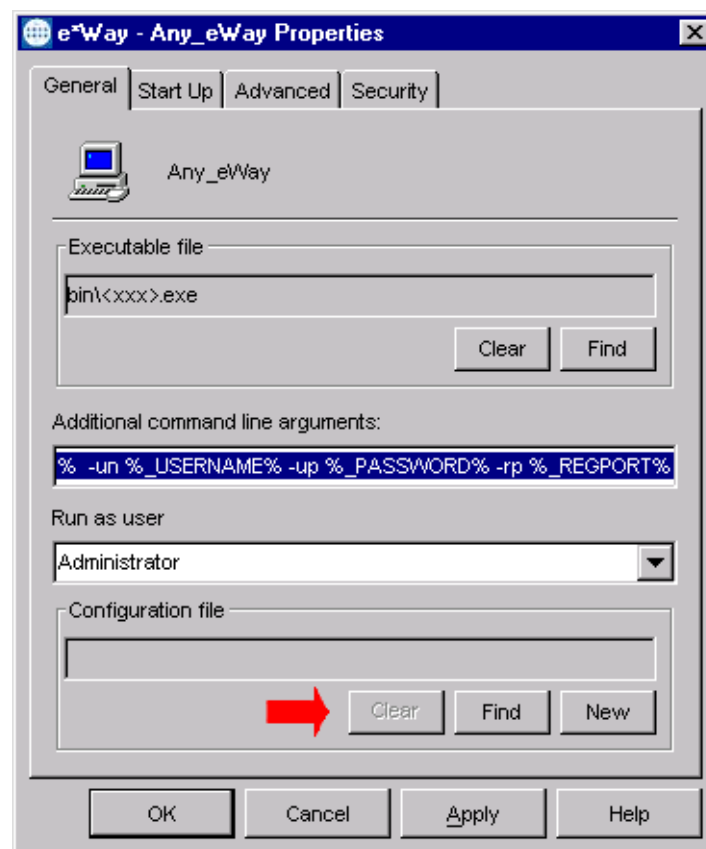
2 Under **Configuration File**, click **New** to create a new file or **Find** to select an existing configuration file. If you select an existing file, an **Edit** button appears; click the button to edit the currently selected file.

3 You are now in the e*Way Configuration Editor.

Using the e*Way Editor

Figure 40   The e*Way Configuration Editor
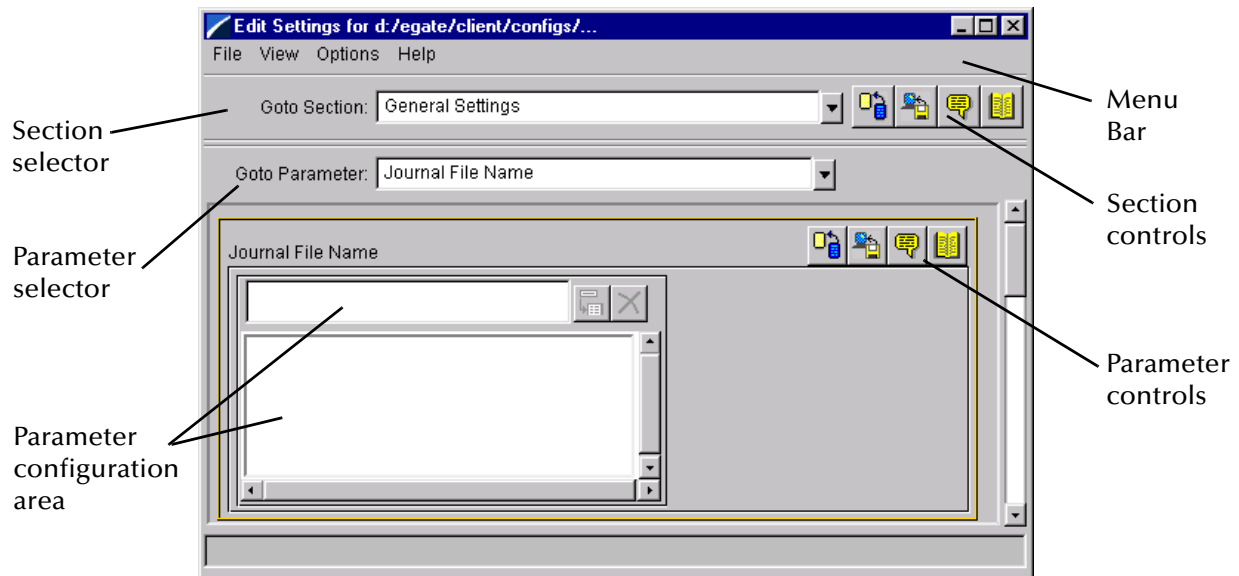
The e*Way Editor controls fall into one of six categories:

- 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 Help system that launches from the Toolbar's Help menu. 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.*

4.2.4 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.

4.2.5 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 41). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 41   e*Way Properties (Start-Up Tab)
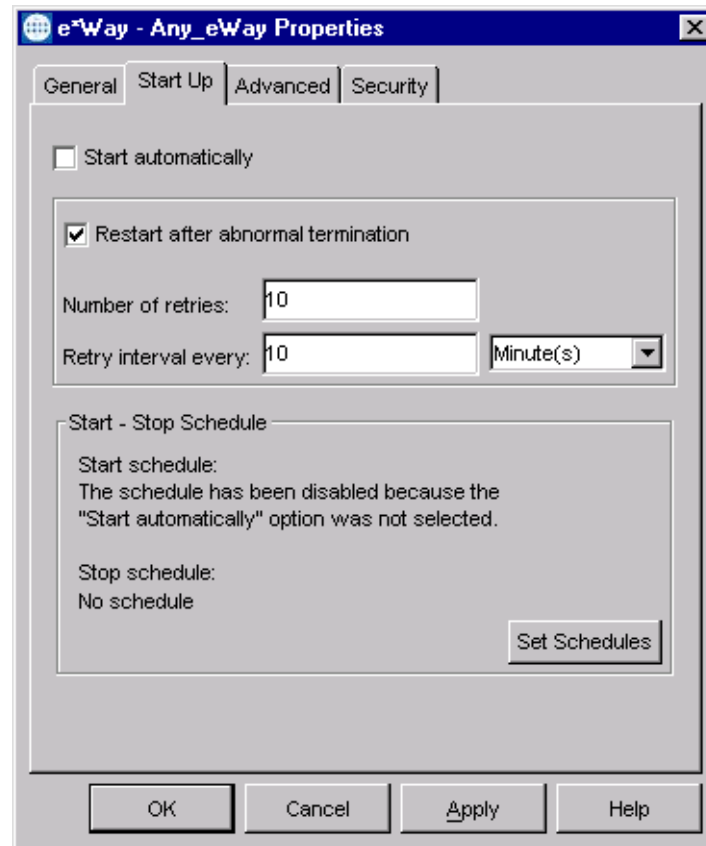
To set the e*Way's startup properties

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**.

4.2.6 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**, and the dialog window appears (see Figure 42).

Figure 42 e*Way Properties (Advanced Tab - Log Option)
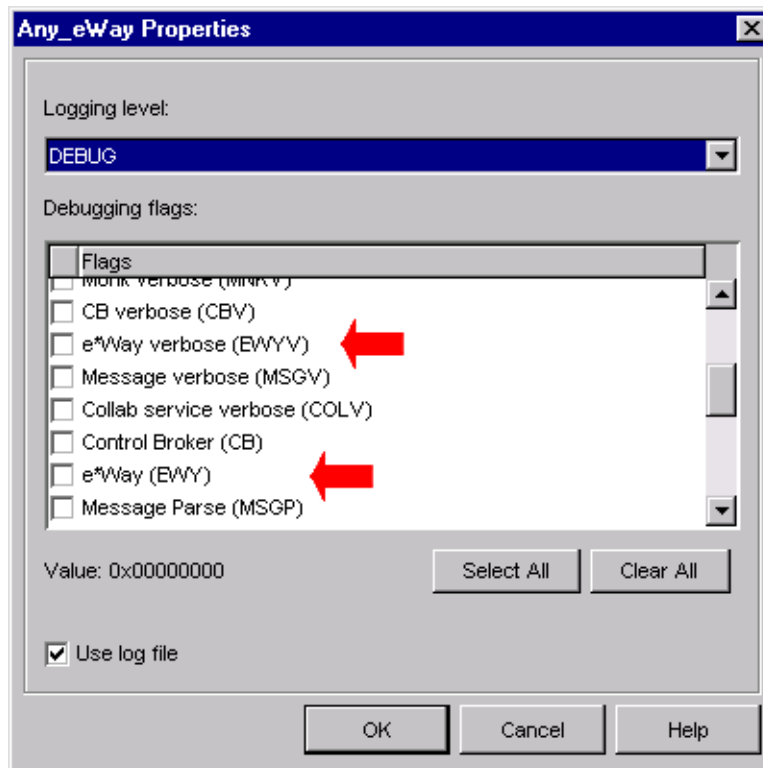
4 Select **DEBUG** for the **Logging level**.

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.

4.2.7 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 routed to the e*Gate Monitor 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 Enterprise Manager

- Does the e*Way have the correct Collaborations assigned?

- Do those Collaborations use the correct Collaboration Services?

- Is the logic correct within any Collaboration Rules script employed by this e*Way's Collaborations?

- Do those Collaborations subscribe to and publish Events appropriately?

- Are all the components that provide information to this e*Way properly configured, and are they sending the appropriate Events correctly?

- Are all the components to which this e*Way sends information 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 environmental variable includes a path to the Siebel Event-Driven dynamically-loaded libraries.

In the Siebel 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 Rules 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 e*Gate monitoring system to monitor operations and performance.

4.3.3 Monk Errors

Monk errors such as the following have been reported when using COM/DCOM:

"MONKEXCEPT:0181: MONK_X_Arg_c_Get: desired element(2) `type' does not `match`"

Running the Microsoft utility **Regclean.exe** to clean up corrupted registry files appears to correct this problem.

Operational Overview

This chapter contains an overview of Siebel-e*Way interface and the architecture and basic internal processes of the Siebel Event-Driven e*Way.

5.1 Interacting with Siebel

5.1.1 Object Layers

The object definitions in Siebel Enterprise Applications fall into three separate architectural layers (excluding the third-party DBMS), as shown in Figure 43.
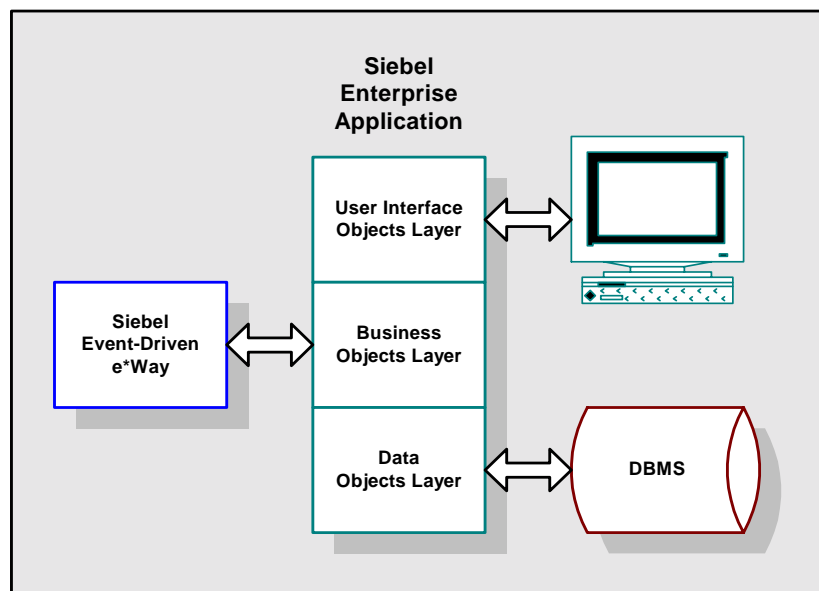
Figure 43 Siebel Object Layers

Business Objects Layer

The Business Object layer is the layer with which the Siebel Event-Driven e*Way interacts. It consists of Business Object Definitions, which are built on Data Object Definitions, and selectively combine and associate Data Object Definitions into logical

data constructs that are useful for application design. Two of these logical constructs, for example, are Business Components (record structures comprised of Columns from multiple joined Tables) and one-to-many Links between record structures.

5.1.2 e*Gate to Siebel

Process Flow

Figure 44 e*Gate-to-Siebel Process Flow
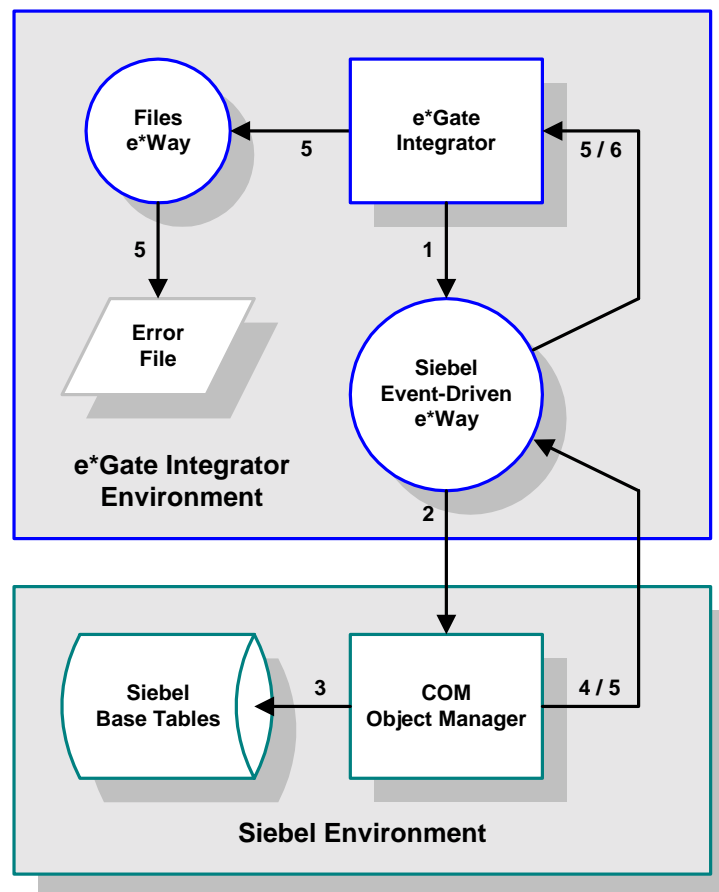
1 The Siebel Event-Driven e*Way receives data corresponding to a business event in the source application, in the form of message definitions within an ETD.

2 It then invokes the Siebel COM Object Manger.

3 The Siebel COM Object Manger loads the business event into the Siebel Base Tables, one message definition at a time.

4 After the COM Object Manager successfully processes each message definition, a commit to the Siebel database is issued. Once all the message definitions in the

message are successfully processed, an **ACK** is sent back to the e*Way, allowing the next message to be submitted for processing.

5   If a single message definition is not successfully processed, the entire inbound message is failed and written to an error file by a Files e*Way. To prevent the message from being resent to the Siebel Event-Driven e*Way, an **ACK** is sent to e*Gate to remove the message from the queue. Also within the failed message, a pointer is saved that identifies the exact message definition that failed.

6   User intervention is required to edit the error file and correct the offending data. This can be accomplished using a standard file editor.

Figure 45   Error-Correction Process Flow
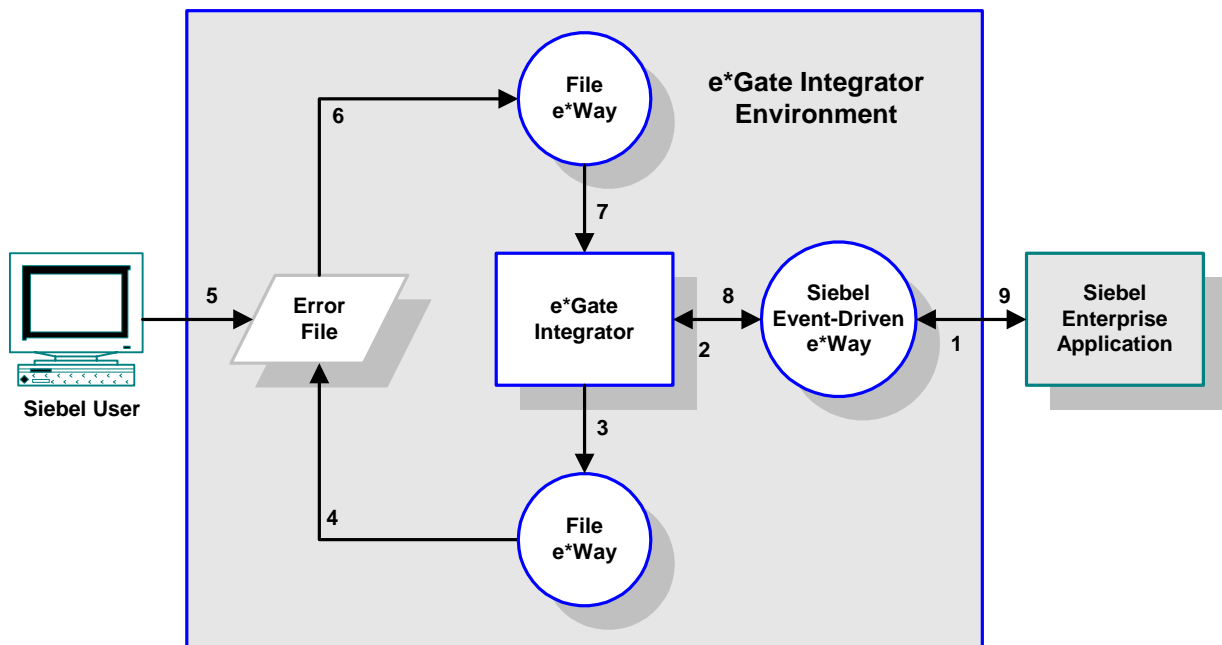
Figure 45 shows the process flow for error correction. The numbers indicate the order of the processing flow, and do not correspond to the numbers associated with Figure 44.

Transaction Management

The interface allows the transactions to be managed down to the smallest logical unit of work-a single message definition. When an error occurs loading the data with COM, the entire message structure, which could contain multiple message definitions, is written to a general error file along with a flag that indicates the nature of the failure. This allows a user to correct the offending data and resubmit the entire message structure. The flag prevents the reprocessing of message definitions that already have been processed successfully.

Figure 46 e*Gate-to-Siebel Transaction Management
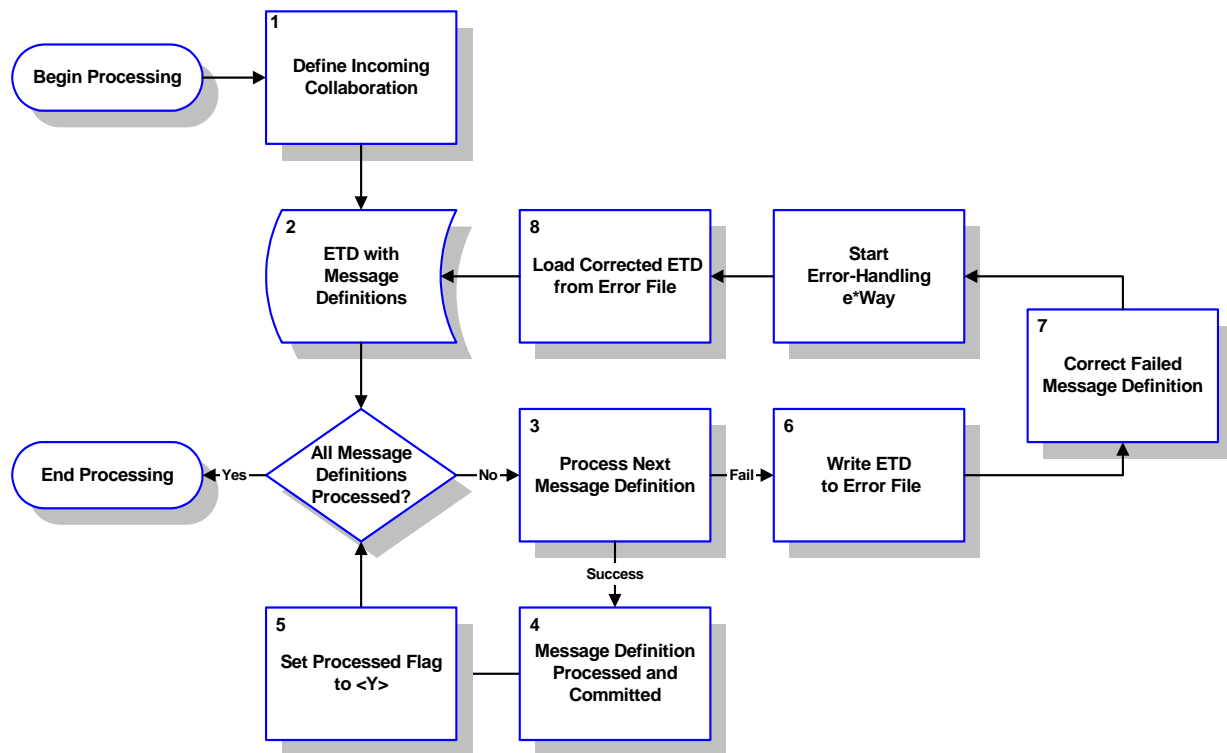
The processing of an Event Type Definition for a single business event by the Siebel Event-Driven e*Way is shown in Figure 46. The processing steps are as follows.

1 When a business event occurs in the source system, the data from the transaction is introduced to the Siebel Event-Driven e*Way through a Collaboration.

2 The result is an ETD that incorporates of one or more message definitions containing the translated data for loading into Siebel.

3 Each message definition within the ETD is submitted to the e*Way, one at a time.

4 A separate commit is issued after the successful processing of each message definition using the Siebel Event-Driven e*Way **Process Outgoing Message Function** parameter.

5 These steps are repeated until all the message definitions in the ETD have been successfully processed. Once the entire ETD is processed, an **ACK** is sent to the e*Gate by the Siebel Event-Driven e*Way to signify that it is ready for the next Event.

6 If a message definition fails, the entire ETD is failed and written to a general error file by an error-handling Files e*Way. An **ACK** is sent to e*Gate to prevent the message structure from being resent to the e*Way.

7 The failed message definition can be corrected manually by using a file editor.

8 Once the failed message definition is corrected, the error file is moved to the error-handling directory where another Files e*Way automatically loads and reprocesses the corrected error file or ETD.

9 The processed flag indicates the message definitions that have not yet been successfully committed to the database. This prevents any message definitions that were successfully processed prior to the failure from being reprocessed by the Files e*Way.

Application Logic & Business Rules

As each message definition is being processed, the COM Data Server enforces the same application logic and business rules as if the transaction were taking place online. This includes all screen, views, edits, and security checks. Using one of the Siebel-supported object models ensures that inserts, updates, and deletes are processed by the transaction processor and that changes are propagated to Siebel remote users.

5.1.3 **Siebel to e*Gate**

Process Flow

Figure 47 Siebel-to-e*Gate Process Flow
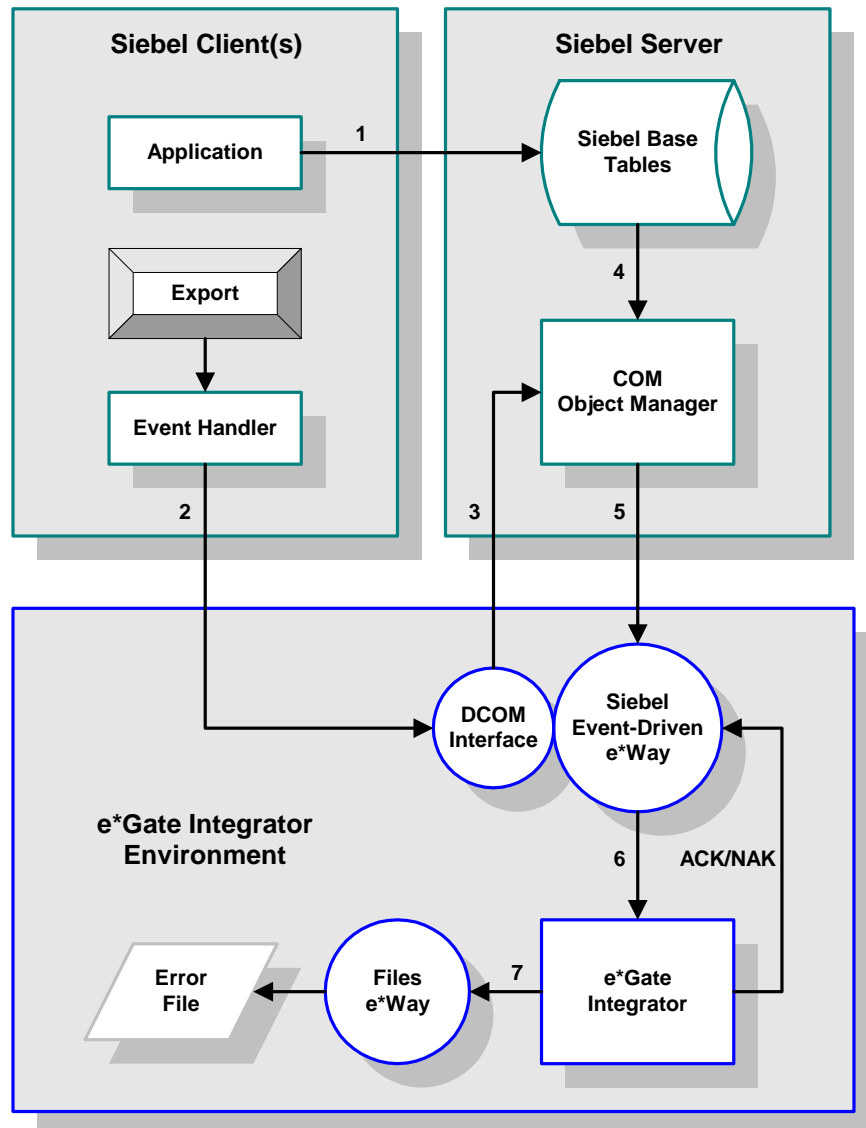
1 When a business event occurs in Siebel, the transaction is posted to the Siebel Base Tables.

2 The Siebel Event Handler triggers the e*Way's DCOM Interface and passes a message string to it. The message string represents key values for the newly-created record, and is well-defined to encode field name(s), value(s), and logical operator(s)—for example, **Name-LIKE-Account**. (A custom push button can be added to the Siebel Application GUI to initiate this action.)

3   The message string is passed to a lookup function, which interacts with the Siebel COM interface.

4   The Siebel COM Object Manager then extracts the requested data from the Siebel Base Tables.

5   The Object Manager then populates the e*Gate ETD for the Siebel business component with the newly-created record.

6   The data is passed to e*Gate for routing to the target e*Way and translation to the target application's ETD. Once an **ACK** is received from e*Gate, the customized status field for the business component is switched to the **Completed** state.

7   Unsuccessfully-processed data is written to an error file by a Files e*Way, and e*Gate sends a **NAK** to the Event-Driven e*Way.

Data Extraction

Triggered Mode

A user of a Siebel Client initiates the data extraction process by means of an export feature (such as a push button) on the Siebel GUI. Siebel then instantiates the DCOM server in the Siebel-outbound e*Way, which uses a lookup function to fetch data from Siebel.

Scheduled Mode

The e*Way also has an optional, periodic lookup function registered to process any possible unprocessed records in a pre-defined interval. Aside from an additional status-field search capability, the implementation of this lookup function is very similar to the invoked function. This alternative, active-polling, mode is provided by **SiebelComQueryResponse** template.

Application Logic & Business Rules

As each message definition is being processed, the COM Object Manager enforces the same application logic and business rules as if the transaction were taking place online. This includes all screen, views, edits, and security checks. The importance of using the one of the Siebel-supported object models is to ensure inserts, updates, and deletes are processed by the transaction processor and changes propagated to Siebel Remote Users.

5.2 e*Way Architecture

Conceptually, the Siebel Event-Driven e*Way can be viewed as a three-layered structure (see Figure 48). 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 48   Siebel Event-Driven e*Way Architecture
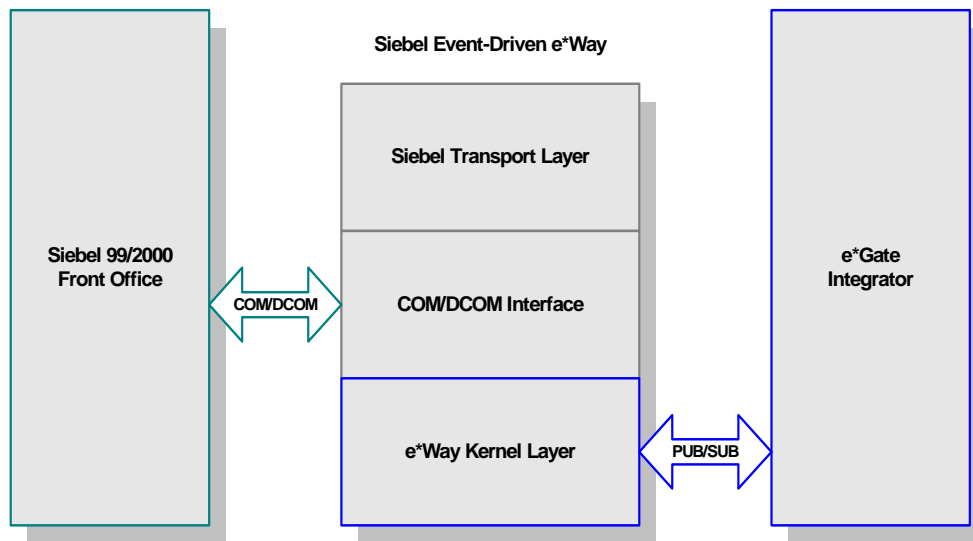
The upper layers of the e*Way use Monk functions to perform Business Process modeling and ETD mapping, package data as e*Gate *Events*, send those Events to Collaborations, and manage interaction with the external system. These layers are built upon an e*Way Kernel layer that manages the basic operations 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.

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 49. Any Collaboration that publishes its processed Events only to an external system does *not* require *any* IQs.

Figure 49   Collaborations and IQs
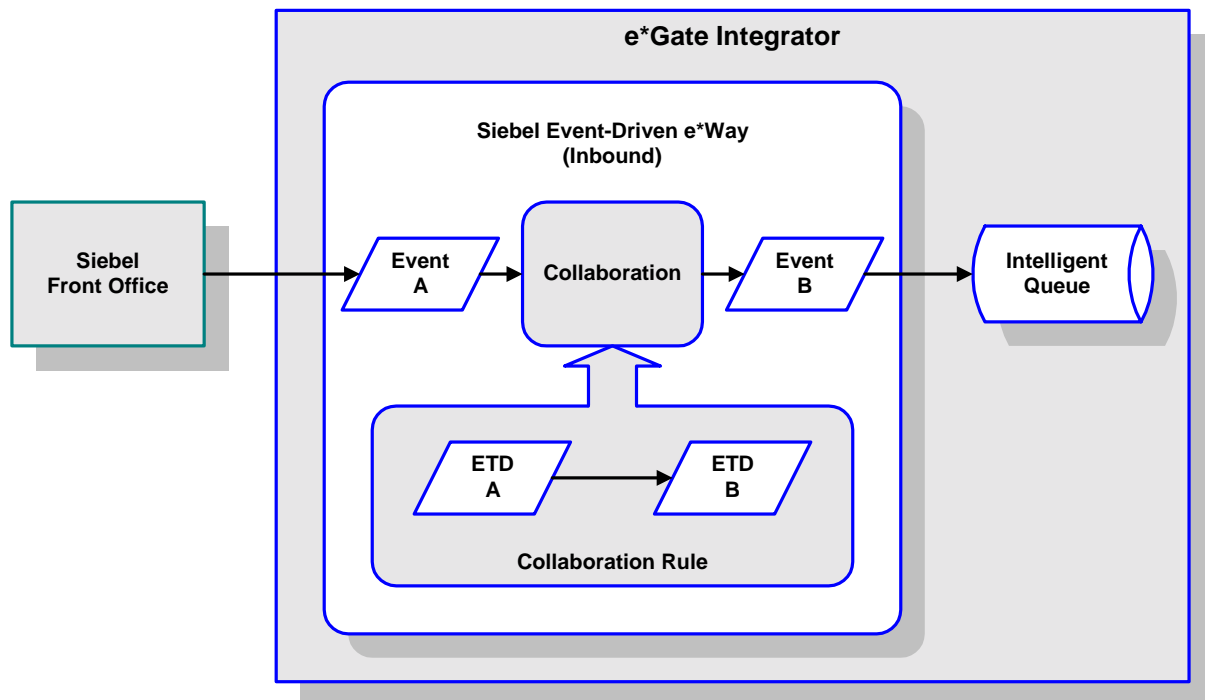
Configuration options that control the Monk environment and define the Monk functions used to perform various e*Way operations are discussed in **Chapter 6**. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft Word* or *Notepad*). 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*.

5.3 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 the*
*Siebel Event-Driven e*Way.*

The most basic processes carried out by an e*Way are listed in the following diagram. 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.

Table 11 Basic e*Way Processes

Process	Monk Configuration Sections
e*Way Initialization	**Startup Function** on page 92 (also see **Monk Environment Initialization File** on page 92)
Connection to External System	**External Connection Establishment Function** on page 95 **External Connection Verification Function** on page 95
Data Exchange	**Event-driven Data Exchange** **Process Outgoing Message Function** on page 93 **Schedule-driven Data Exchange** **Exchange Data with External Function** on page 94 **Positive Acknowledgment Function** on page 96 **Negative Acknowledgment Function** on page 97
Disconnection from External System	**External Connection Shutdown Function** on page 96
e*Way Shutdown	**Shutdown Command Notification Function** on page 98

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

Figure 50   Initialization Process

```
                    ╭──────────────╮
                    │  Start e*Way │
                    ╰──────────────╯
                            │
                            ▼
              ┌───────────────────────────┐
              │            Load           │
              │   Monk Initialization     │
              │            file           │
              └───────────────────────────┘
                            │
                            ▼
              ┌───────────────────────────┐
              │  Execute any Monk function│
              │   having the same name as │
              │    the initialization file│
              └───────────────────────────┘
                            │
                            ▼
              ┌───────────────────────────┐
              │                           │
              │     Load Startup file     │
              │                           │
              └───────────────────────────┘
                            │
                            ▼
              ┌───────────────────────────┐
              │  Execute any Monk function│
              │   having the same name as │
              │       the startup file    │
              └───────────────────────────┘
```
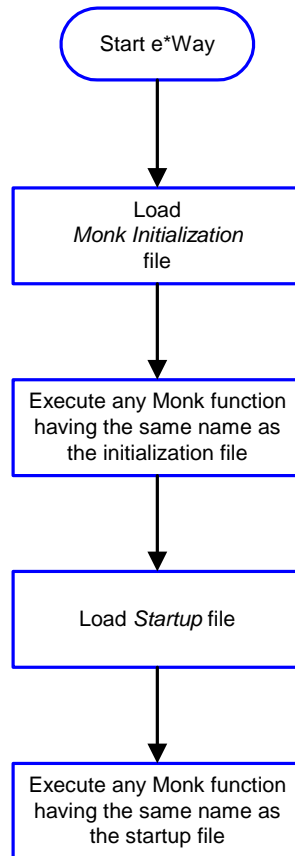
Connect to External Process

Figure 51 illustrates how the e*Way connects to the external system, using the **External Connection Establishment Function** and **External Connection Verification Function**.

Figure 51   Connection Process
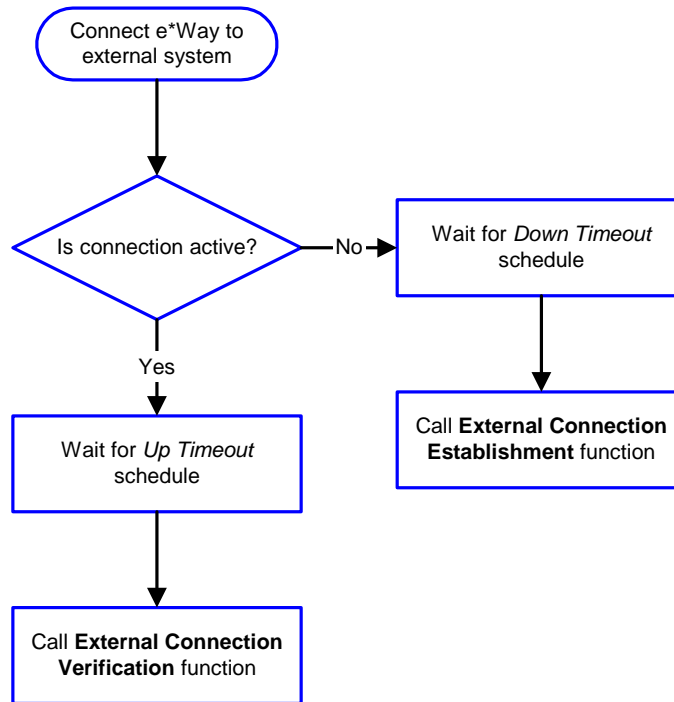
Note: *The e*Way selects the connection function based on an internal **up/down** flag rather than a poll to the external system. See **Figure 53 on page 82** and **Figure 52 on page 81** for examples of how different functions use this flag.*

*User functions can manually set this flag using Monk functions. See **send-external-up** on page 138 and **send-external-down** on page 138 for more information.*

Data Exchange Process

Event-driven

Figure 52 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 52 Event-Driven Data Exchange Process
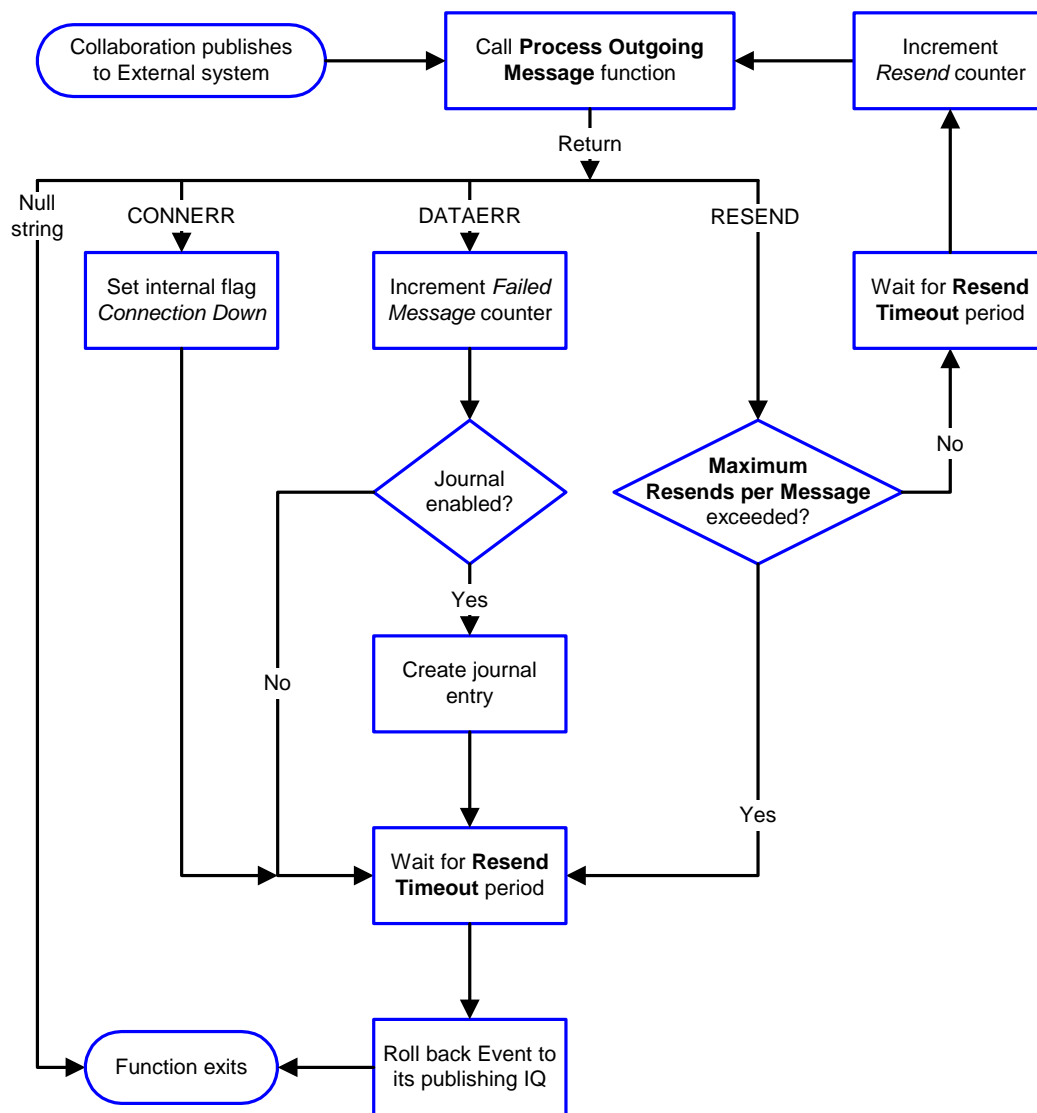
Schedule-driven

Figure 53 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 53 Schedule-Driven Data Exchange Process
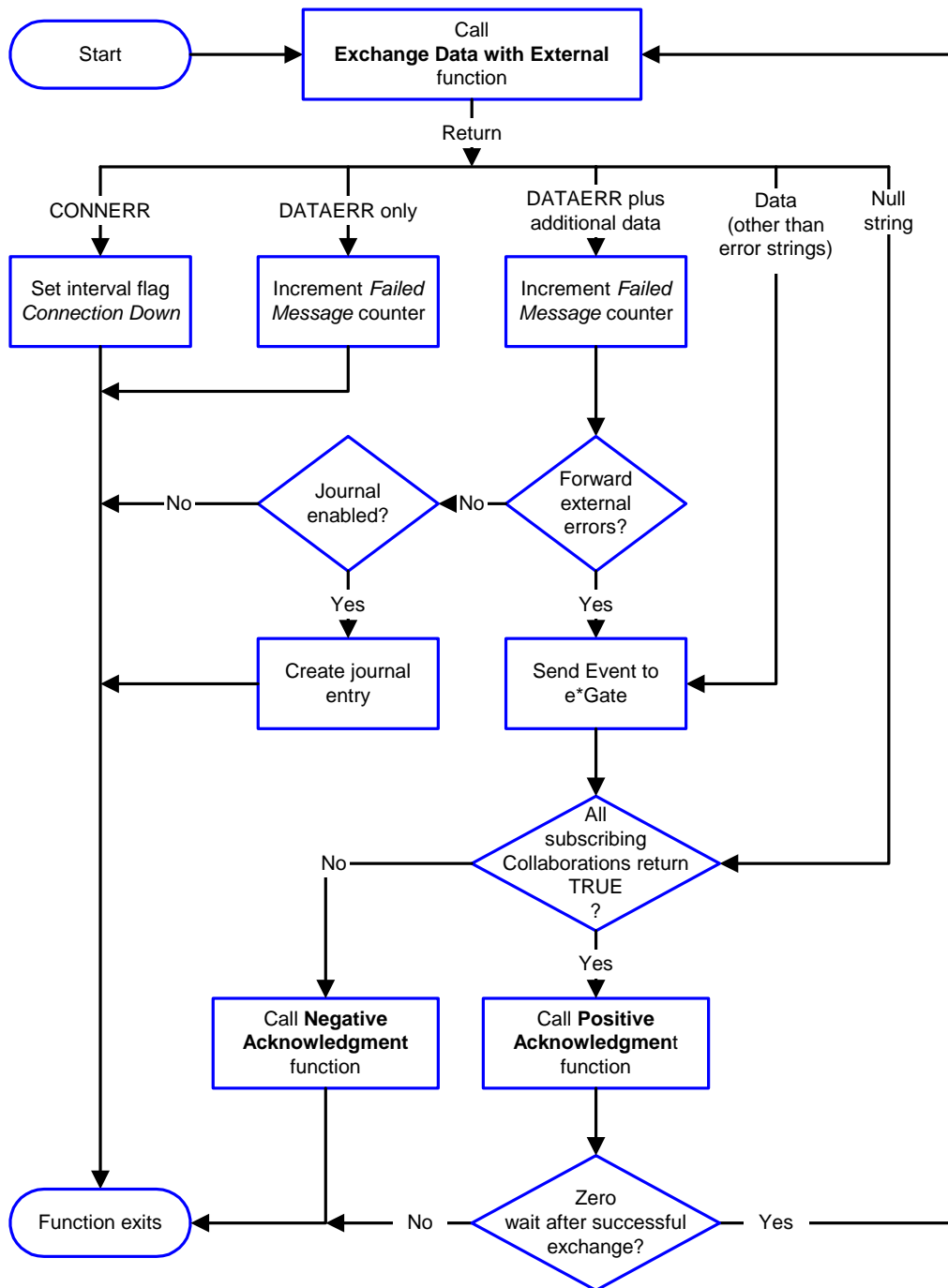
Start can occur in any of the following ways:

- *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 54):

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

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

Figure 54   Send Event to e*Gate with Confirmation
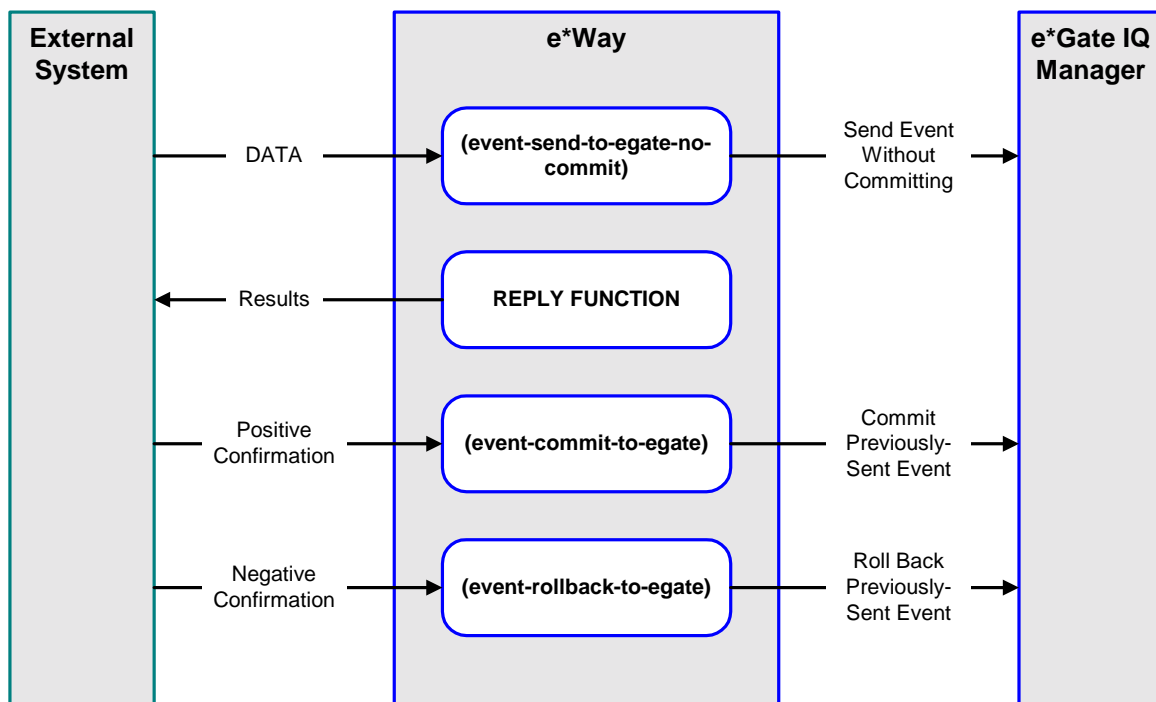
After the function exits, the e*Way waits for the next *Start* time or command.

Disconnect from External Process

Figure 55 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

Figure 55 Disconnect Process

```
          ┌─────────────────────────┐
          │   Control Broker issues  │
          │    Suspend command       │
          └─────────────────────────┘
                       │
                       ▼
       ┌────────────────────────────────────┐
       │ Call External Connection Shutdown   │
       │      function                       │
       │ with SUSPEND_NOTIFICATION parameter │
       └────────────────────────────────────┘
                       │
                 Return any value
                       ▼
          ┌─────────────────────────┐
          │  e*Way closes connection │
          └─────────────────────────┘
```
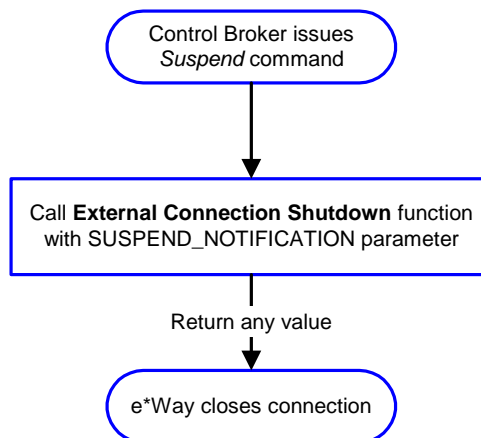
Shutdown Process

Figure 56 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

Figure 56 Shutdown Process

```
        ┌──────────────────────┐
        │ Control Broker issues │────(Optional)────┐
        │   Shutdown command    │                  │
        └──────────────────────┘                  │
                  │                                │
                  ▼                                ▼
  ┌────────────────────────────────┐   ┌────────────────────────┐
  │ Call Shutdown Notification      │   │ Call waiting-to-shutdown│
  │ function                        │   │      function           │
  │ with SHUTDOWN_NOTIFICATION      │   └────────────────────────┘
  │ parameter                       │              │
  └────────────────────────────────┘              ▼
           │                           ┌────────────────────────┐
         Return                        │ Inform External system │
    ┌──────────┴─────────┐             │ that Shutdown command  │
 Null string or   any other value      │ has been issued        │
 "SUCCESS"              │               └────────────────────────┘
    │                   ▼
    │          ┌─────────────────┐
    │          │    Wait for      │
    │          │ shutdown-request │
    │          │    function      │
    │          └─────────────────┘
    └─────────┬─────────┘
              ▼
     ┌─────────────────┐
     │ e*Way shuts down │
     └─────────────────┘
```
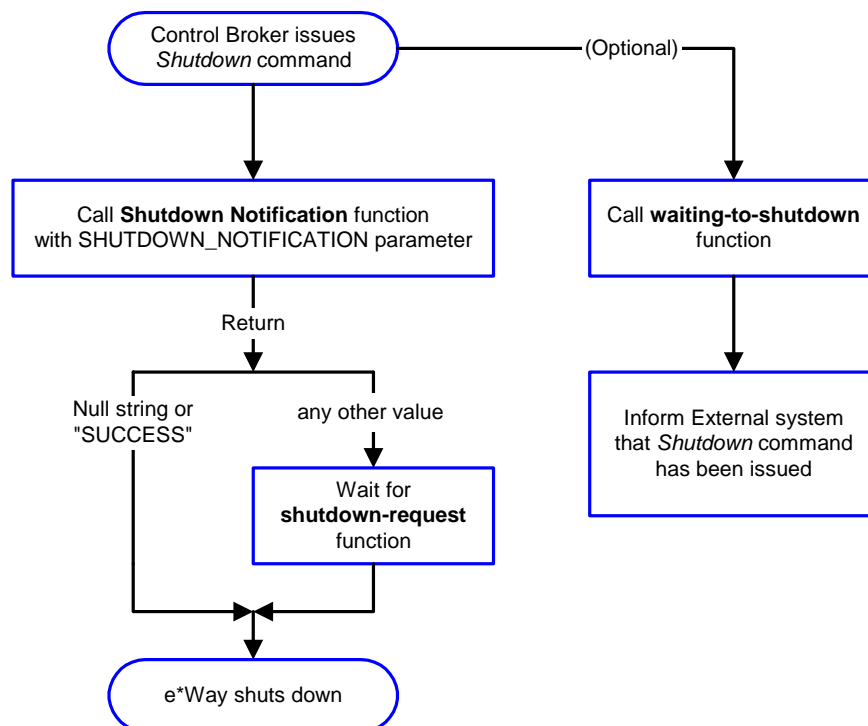
Configuration Parameters

This chapter describes the configuration parameters for the Siebel Event-Driven e*Way.

6.1 Overview

The e*Way's configuration parameters are set using the e*Way Editor; see **Configuring the e*Way** on page 59 for procedural information. The default configuration is provided in **stcewsiebeleventdriven.def**. The Siebel Event-Driven e*Way's configuration parameters are organized into the following sections:

General Settings on page 86

Communication Setup on page 88

Monk Configuration on page 91

Siebel Setup on page 99

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

The Journal file is used 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 between **1** and **1,024**. The default is **5**.

Max Failed Messages

Description

Specifies the maximum number of failed messages 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 between **1** and **1,024**. The default is **3**.

Forward External Errors

Description

Selects whether or not error messages received from the external system that begin with the string **"DATAERR"** are queued to the e*Way's configured queue.

Required Values

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

See also

Exchange Data with External Function on page 94

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 parameters that are set within the e*Way Editor are independent of those set within the e*Gate Enterprise Manager. If you choose to operate this e*Way on a schedule, be sure that you define compatible schedules in both the e*Way Editor and the e*Gate Enterprise Manager.*

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

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

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

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

If you do not do so, the e*Way terminates execution when the schedule attempts to start.

Additional Information

When the schedule starts, the e*Way determines whether or not it is waiting to send an **ACK** or **NAK** to the external system (using the Positive and Negative Acknowledgment functions) and whether or not 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 interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Since months do not all contain equal numbers of days, be sure not to provide boundaries that would cause an invalid date selection (i.e. the 30th of every month would not include February).

Exchange Data Interval

Description

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

Required Values

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

Additional Information

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

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

See **Down Timeout** and **Stop Exchange Data Schedule** for additional information about the data-exchange schedule.

Down Timeout

Description

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

Required Values

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

Up Timeout

Description

Specifies the number of seconds the e*Way waits between calls to the **External Connection Verification Function**.

Required Values

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

Resend Timeout

Description

Specifies the number of seconds the e*Way waits between attempts to resend an message to the external system, after receiving an error message.

Required Values

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

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**.

If this parameter is set to **Yes**, the e*Way immediately invokes the **Exchange Data with External Function** if the previous exchange function returned an Event.

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**.

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. There is no default value for this parameter.

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

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

Required Values

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

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

Additional information

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

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which is loaded after the auxiliary library directories are loaded. Use this feature to initialize any Monk variables that are used by the e*Way's function scripts.

Required Values

A filename within the **Load Path**, or a filename plus path information (relative or absolute). If path information is specified, that path is appended to the **Load Path**. The default is **siebel-eventdriven-init**.

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

Additional information

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

Startup Function

Description

Specifies a Monk function that the e*Way loads and invokes upon startup or whenever the e*Way's configuration is reloaded. This function should be used to initialize the external system before data exchange starts.

Required Values

The name of a Monk function or the name of a file containing a Monk function. There is no default value for this parameter.

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

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

Process Outgoing Message Function

Description

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

Required Values

The name of a Monk function, the name of a file containing a Monk function. There is no default value for this parameter.

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 Enterprise Manager).

- 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 called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven).

Required Values

The name of a Monk function, the name of a file containing a Monk function. There is no default value for this parameter.

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

Returns

- A *null string* ("") indicates that the data exchange was completed successfully, but with no resultant data sent back to the e 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**. Queueing, 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 to establish (or re-establish) a connection to the external system. This function is executed according to the interval specified within the **Down Timeout** parameter, and is *only* called according to this schedule. The **External Connection Verification Function** (see below) is called when the e*Way has determined that its connection to the external system is up.

Required Values

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

- For Incoming e*Ways (Siebel-to-e*Gate), the default is **siebel-eventdriven-dummy**.

- For Outgoing e*Ways (e*Gate-to-Siebel), the default is **siebel-eventdriven-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 to confirm that the external system is operating and available. This function is executed according to the interval specified within the **Up Timeout** parameter, and is *only* called according to this schedule. The **External Connection Establishment Function** (see previous) is called when the e*Way has determined that its connection to the external system is down.

Required Values

The name of a Monk function or the name of a file containing a Monk function. If nothing is specified, the e*Way executes the **External Connection Establishment Function** in its place. The default is **siebel-eventdriven-verify-connect**.

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

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

External Connection Shutdown Function

Description

Specifies a Monk function that the e*Way calls to shut down the connection to the e*Way. 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. There is no default value for this parameter.

Note: *This parameter is optional and may 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

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default is **siebel-eventdriven-ack**.

Note: *This parameter is conditional, and must be specified only if the **Exchange Data with External Function** is defined.*

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. If the Event's processing is completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the **Positive Acknowledgment Function** (otherwise, the e*Way executes the **Negative Acknowledgment Function**).

- 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

Specifies a Monk function that the e*Way calls when the e*Way fails to process and queue Events from the external system.

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default is **siebel-eventdriven-nack**.

Note: *This parameter is conditional, and must be specified only if the **Exchange Data with External Function** is defined.*

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 only called during the processing of inbound Events. After the **Exchange Data with External Function** returns a string that is transformed into an

inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is not completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the **Negative Acknowledgment Function** (otherwise, the e*Way executes the **Positive Acknowledgment Function**).

- 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

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. There is no default value for this parameter.

Note: *This parameter is optional and may 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 Siebel Setup

The parameters in this section help you set up the information required by the e*Way to communicate with the Siebel Server.

Communication Direction

Description

The direction of data flow (e*Gate to Siebel, Siebel to e*Gate).

Required Values

One of the following options:

- Outbound Toward Siebel
- Inbound From Siebel
- Both

Siebel Login Name

Description

The user name for the Siebel server.

Required Values

A valid user name.

Siebel Login Password

Description

The password for the Siebel server.

Required Values

The password corresponding to the user name entered above.

Siebel Config File

Description

The location of the Siebel configuration file.

Required Values

Directory and file name.

Additional information

The Siebel configuration file specifies a default Siebel server to which to connect, in addition to other servers. Make sure that the server you need to connect to is set as the default. To connect to more than one server, you need multiple Siebel configuration files.

Siebel Business Object

Description

The Siebel object type that is being created or queried on the remote Siebel system.

Required Values

The name of a valid Siebel Business Object.

API Functions

This chapter describes the various Monk functions used by the SeeBeyond e*Way Intelligent Adapter for Siebel (Event-Driven).

7.1 Overview

The Siebel Event-Driven e*Way's functions are categorized as follows:

- **Siebel Transport Functions** on page 102
- **Siebel General Functions** on page 124
- **Example Functions** on page 131
- **Generic e*Way Functions** on page 134

7.2 Siebel Transport Functions

These Monk APIs are developed specifically to control interactions between the Siebel Event-Driven e*Way and a Siebel application.

sieb-associate on page 103

sieb-error on page 103

sieb-get-associate-bc on page 104

sieb-get-mvg-bc on page 105

sieb-get-picklist-bc on page 106

sieb-get-search-expr on page 106

sieb-query on page 107

sieb-query2 on page 108

sieb-select-picklist-fields on page 108

sieb-select-mvg-fields on page 110

sieb-struct-delete on page 111

sieb-struct-delete2 on page 111

sieb-struct-get-bc on page 112

sieb-struct-insert on page 113

sieb-struct-insert2 on page 113

sieb-struct-insert-with-pick on page 114

sieb-struct-lookup on page 115

sieb-struct-lookup2 on page 116

sieb-struct-set-field on page 117

sieb-struct-set-field2 on page 118

sieb-struct-single-insert on page 118

sieb-struct-single-insert2 on page 119

sieb-struct-update on page 119

sieb-struct-update2 on page 120

sieb-struct-write on page 121

sieb-struct-write2 on page 121

sieb-struct-write-pick-mvg on page 122

sieb-associate

Description

Creates a many-to-many relationship between parent and child business component.

Signature

```
(sieb-associate <bus-com-hdl> <query-cond>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
query-cond	Vector	Query conditions.

Returns

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

Throws

None.

Location

sieb-associate.monk

Examples

```
(set! ret (sieb-associate <bus-com-hdl> <query-cond>))
(if ret
      (display "OK")
      (display "Failed")
)
```

sieb-error

Description

Tests **param-vec** for error messages and displays them to the current output port.

Signature

```
(sieb-error <param-vec>)
```

Parameters

Name	Type	Description
param-vec	Vector	Invoke parameter vector.

Returns

If no error is found, a Boolean true (**#t**); otherwise, an error string.

Throws

None.

Location

sieb-error.monk

Examples

```
(com-invoke <siebobj-handle> "LoadObjects" "FUNC" <param-vec>)
(if (sieb-error <param-vec>)
     (display "OK")
     (display "Failed"
)
```

sieb-get-associate-bc

Description

Obtains the association business component handle.

Signature

```
(sieb-get-associate-bc <bus-com-hdl>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.

Returns

If successful, the business component handle (**bus-com-hdl**) associated with **sieb-get-associate-bc**. Upon failure, a Boolean false (**#f**).

Throws

None.

Location

sieb-get-assoc-bc.monk

Examples

```
(set! assoc-bus-com-hdl (sieb-get-associate-bc <bus-com-hdl>))
```

sieb-get-field-values

Description

This function returns a vector containing the value or values for a **<bc-field>** that matches the query vector for the business component specified by **<bc-name>**.

Signature

```
(sieb-get-field-values <bc-name> <bc-field> <query-vec>)
```

Parameters

Name	Type	Description
bc-name	Monk object	The name of the business component to get the values from. (This business component must exist on the Business Object selected in this e*Way's configuration file.)
bc-field	Monk object	The name of the field to retrieve values for.
query-vec	Vector	Search vector of three strings in the format: (vector <field-name> <operator> <condition>).

Returns

A vector containing the values for the **bc-field** from the record(s) returned from the query condition(s).

Throws

None.

Location

sieb-get-field-values.monk

Examples

```
(set! assoc-bus-com-hdl (sieb-get-field-values <bc-name> <bc-field>
        <query-vec>))
```

sieb-get-mvg-bc

Description

Replaces an *association* business component with a *multiple value group* business component.

Signature

```
(sieb-get-mvg-bc <bus-com-hdl> <field-name>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
field-name	String	The name of the field on the business component that contains the MVG.

Returns

If successful, the business component handle (**bus-com-hdl**) associated with **sieb-get-mvg-bc**. Upon failure, a Boolean false (**#f**).

Throws

None.

Location

sieb-get-mvg-bc.monk

Examples

```
(set! assoc-bus-com-hdl (sieb-get-mvg-bc <bus-com-hdl> <field-name>))
```

sieb-get-picklist-bc

Description

Replaces an *association* business component with a *picklist* business component.

Signature

```
(sieb-get-picklist-bc <bus-com-hdl> <field-name>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
field-name	String	The name of the field on the business component that contains the Pick List.

Returns

If successful, the business component handle (**bus-com-hdl**) associated with **sieb-get-picklist-bc**. Upon failure, a Boolean false (**#f**).

Throws

None.

Location

sieb-get-picklist-bc.monk

Examples

```
(set! assoc-bus-com-hdl (sieb-get-picklist-bc <bus-com-hdl> <field-
      name>))
```

sieb-get-search-expr

Description

Generates a query condition vector from the message **expr-msg**.

Signature

```
(sieb-get-search-expr <expr-msg>)
```

Parameters

Name	Type	Description
expr-msg	String	**sieb-query-expr** structure message.

Returns

A search expression vector (see example below).

Throws

None.

Location

sieb-get-search-expr.monk

Examples

```
(sieb-get-search-expr "Product
    Line~=~Toothpaste|Product~=~TeethSoBright|Price~>~4.99|")
```

 returns:

```
(vector (vector 'Product Line' '=' 'Toothpaste') (vector 'Product' '='
    'TeethSoBright') (vector 'Price' '>' '4.99'))
```

sieb-query

Description

Invokes Siebel object methods to query against Siebel database. Note that the handle to business component object has to be reset before calling **sieb-query**. SeeBeyond recommends that the function **sieb-get-search-expr** be called to generate the query condition from a simple string.

Signature

```
(sieb-query <query-cond>)
```

Parameters

Name	Type	Description
query-cond	Vector	Query condition vector.

Returns

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

Throws

None.

Location

sieb-query.monk

Examples

```
(sieb-query (sieb-get-search-expr "Name~LIKE~STC*|"))
```

sieb-query2

Description

Invokes Siebel object methods to query against Siebel database. Note that the handle to business component object has to be reset before calling **sieb-query2**. SeeBeyond recommends that the function **sieb-get-search-expr** be called to generate the query condition from a simple string.

Signature

```
(sieb-query2 <bus-com-hdl> <query-cond>)
```

Parameters

Name	Type	Description
query-cond	Vector	Query condition vector.
bus-com-hdl	Monk object	Business component handle.

Returns

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

Throws

None.

Location

sieb-query2.monk

Examples

```
(sieb-query2 bus-com-hdl
      (sieb-get-search-expr "Name~LIKE~STC*|"))
```

sieb-select-picklist-fields

Description

Allows you to select values for a picklist on a business component specified by **bc-handle**.

Signature

```
(sieb-select-picklist-fields <bc-handle> <picklist-vec>)
```

Parameters

Name	Type	Description
bc-handle	Monk object	The handle of the business component containing the picklist fields. It must already have *one* record selected. This means you must perform a query using this **bc-handle** and get the **FirstRecord** from the query before the handle is passed to this function.
picklist-vec	Vector	Picklist vector (see Additional Information, below)

Returns

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

Throws

None.

Location

sieb-select-picklist-fields.monk

Additional Information

The Picklist and Multi-value group vectors have the following form:

```
(vector (vector <field-name> <search-expr>) (vector <field-name>
      <search-expr>) ... )
```

where:

Name	Type	Description
field-name	String	The name of the field on the BC that has either the pick list or multi value group.
search-expr	String	The search condition for the picklist or MVG business component. This condition should return one record only.

Examples

picklist-vec:

```
(set! picklist-vec (vector (vector "Unit of Measure" "Name~=~Each|")))
```

sieb-select-mvg-fields

Description

Checks whether or not the records matching the key values exist. If they do, then the function conducts an update; otherwise, an insert. **Keys-rec** specifies a list of keys for searching. Key values are obtained for the business component path node.

Signature

```
(sieb-select-mvg-fields <bc-handle> <mvg-vec>)
```

Parameters

Name	Type	Description
bc-handle	Monk object	The handle of the business component containing the picklist fields. It must already have *one* record selected. This means you must perform a query using this **bc-handle** and get the **FirstRecord** from the query before the handle is passed to this function.
mvg-vec	Vector	Multi-value group vector (see Additional Information, below)

Returns

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

Throws

None.

Location

sieb-select-mvg-fields.monk

Additional Information

The Picklist and Multi-value group vectors have the following form:

```
(vector (vector <field-name> <search-expr>) (vector <field-name>
    <search-expr>) ... )
```

where:

Name	Type	Description
field-name	String	The name of the field on the BC that has either the pick list or multi value group.
search-expr	String	The search condition for the picklist or MVG business component. This condition should return one record only.

Examples

mvg-vec:

```
(set! mvg-vec (vector (vector "Product" (string-append "Id~=~" siebel-
      row-id "|"))))
```

sieb-struct-delete

Description

Searches for records which match the query condition and subsequently deletes them.

Signature

```
(sieb-struct-delete <sieb-obj-node> <query-cond>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the repeating business component object.
query-cond	Vector	Query conditions.

Returns

If successful, returns the Boolean true (**#t**); otherwise, returns false (**#f**).

Throws

None.

Location

sieb-struct-delete.monk

sieb-struct-delete2

Description

Searches for records in the association business components matching the specified business component handle which match the query condition, and subsequently deletes them.

Signature

```
(sieb-struct-delete2 <bus-com-hdl> <sieb-obj-node> <query-cond>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
sieb-obj-node	Path	Path to the repeating business component object.
query-cond	Vector	Query condition.

Returns

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

Throws

None.

Location

sieb-struct-delete2.monk

sieb-struct-get-bc

Description

Obtains the business component handle from a Monk path.

Signature

```
(sieb-struct-get-bc <sieb-obj-node>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the business component node (single instance).

Returns

If successful, the business component handle (**bus-com-hdl**) associated with **sieb-struct-get-bc**; upon failure, a Boolean false (**#f**).

Throws

None.

Location

sieb-struct-get-bc.monk

Examples

```
(define bus-com-hdl
    (sieb-struct-get-bc ~input%sieb-
    account.PROJECT.Account.BUSINESS_COMPONENT.Account)
)
where Account is a repeating field
```

sieb-struct-insert

Description

Calls **sieb-struct-set-field** for all repeating data found on the node.

Signature

```
(sieb-struct-insert <sieb-obj-node>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the business component node.

Returns

None.

Throws

None.

Location

sieb-struct-insert.monk

Examples

```
(sieb-struct-insert
      ~input%sieb-account.PROJECT.Account.BUSINESS_COMPONENT.Account)
```

where Account is a repeating field

sieb-struct-insert2

Description

Calls **sieb-struct-set-field** for all repeating data found on the specified node in the association business components matching the specified business component handle.

Signature

```
(sieb-struct-insert2 <bus-com-hdl> <sieb-obj-node>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
sieb-obj-node	Path	Path to the business component node.

Returns

None.

Throws

None.

Location

sieb-struct-insert2.monk

Examples

```
(sieb-struct-insert2 bus-com-hdl
        ~input%sieb-account.PROJECT.Account.BUSINESS_COMPONENT.Account)
```

where Account is a repeating field

sieb-struct-insert-with-pick

Description

Calls **sieb-struct-set-field** for repeating data found on the node. Unlike **sieb-struct-insert**, this function obtains data in fields that must be picked through a *picklist*.

Signature

```
(sieb-struct-insert-with-pick <sieb-obj-node> <pick-vec>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the business component node.
pick-vec	Vector	Vector having elements composed of field name-search expression pairs (see Additional Information, below).

Returns

None.

Throws

None.

Location

sieb-struct-insert-with-pick.monk

Additional Information

The pick-vec vector has the following form:

```
(vector (vector <field-name> <search-expr>) (vector <field-name>
    <search-expr>) ... )
```

where:

Name	Type	Description
field-name	String	The name of the field on the BC that has either the pick list or multi value group.
search-expr	String	The search condition for the picklist or MVG business component. This condition should return one record only.

Examples

```
(define (pick-vector (make-vector 1 (make-vector 2 ""))))
(vector-set! (vector-ref pick-vector 0) 0 "Product")
(vector-set! (vector-ref pick-vector 0) 1 (string-append "Id~=~" prod-
    rowid "|") )
(if (sieb-struct-insert-with-pick
    ~output%Siebel_Quotes.PROJECT.Quote.BUSINESS_COMPONENT.Quote_Ite
    m pick-vector)
 (begin
  (display (string-append "Processed quote item.")) (newline)
 )
 (begin
  (display (string-append "ERROR: sieb-struct-insert failed for Siebel
    Quote.")) (newline)
 )
)
```

sieb-struct-lookup

Description

Searches the Siebel database according to the query condition, then populates the results (multiple records) onto the passed-in object node.

Signature

```
(sieb-struct-lookup <sieb-obj-node> <query-cond>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the repeating business component node.
query-cond	Vector	Query condition.

Returns

A successful search returns one of the following:

rec-num	Number representing records found and mapped onto the structure.
0	Integer **0** indicating no matching record returned.

An unsuccessful search returns a Boolean false (#f).

Throws

None.

Location

sieb-struct-lookup.monk

Examples

```
(sieb-struct-lookup
     ~sieb-account%sieb-
     account.PROJECT.Account.BUSINESS_COMPONENT.Account
     (sieb-get-search-expr "Name~LIKE~STC*|")
```

sieb-struct-lookup2

Description

Searches the Siebel database according to the query condition in the association business components matching the specified business component handle, then populates the results (multiple records) onto the passed-in object node.

Signature

```
(sieb-struct-lookup2 <bus-com-hdl> <sieb-obj-node> <query-cond>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
sieb-obj-node	Path	Path to the repeating business component node.
query-cond	Vector	Query condition.

Returns

A successful search returns one of the following:

rec-num	Number representing records found and mapped onto the structure.
0	Integer **0** indicating no matching record returned.

An unsuccessful search returns a Boolean false (**#f**).

Throws

None.

Location

sieb-struct-lookup2.monk

Examples

```
(sieb-struct-lookup bus-com-hdl
    ~sieb-account%sieb-
    account.PROJECT.Account.BUSINESS_COMPONENT.Account
    (sieb-get-search-expr "Name~LIKE~STC*|")
```

sieb-struct-set-field

Description

Gets data from **sieb-obj-node** and calls Siebel interface method **Set Field Value**.

Signature

```
(sieb-struct-set-field <sieb-obj-node>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the repeating business component node.

Returns

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

Throws

None.

Location

sieb-struct-set-field.monk

sieb-struct-set-field2

Description

Gets data from **sieb-obj-node** in the association business components matching the specified business component handle, and calls Siebel interface method **Set Field Value**.

Signature

```
(sieb-struct-set-field2 <bus-com-hdl> <sieb-obj-node>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
obj-node	path	Path to the repeating business component node.

Returns

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

Throws

None.

Location

sieb-struct-set-field2.monk

sieb-struct-single-insert

Description

Calls **sieb-struct-set-field** for the data found on the node. Unlike **sieb-struct-insert**, this function is *not* designed to handle repeating nodes.

Signature

```
(sieb-struct-single-insert <sieb-obj-node>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the business component node.

Returns

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

Throws

None.

Location

sieb-struct-single-insert.monk

Examples

```
(sieb-struct-single-insert
     ~input%sieb-
     account.PROJECT.Account.BUSINESS_COMPONENT.Account[0])
```

sieb-struct-single-insert2

Description

Calls **sieb-struct-set-field** for the data found on the specified node in the association business components matching the specified business component handle. Unlike **sieb-struct-insert**, this is *not* designed to handle repeating nodes.

Signature

```
(sieb-struct-single-insert2 <bus-com-hdl> <sieb-obj-node>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
sieb-obj-node	Path	Path to the business component node.

Returns

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

Throws

None.

Location

sieb-struct-single-insert2.monk

Examples

```
(sieb-struct-single-insert bus-com-hdl
     ~input%sieb-
     account.PROJECT.Account.BUSINESS_COMPONENT.Account[0])
```

sieb-struct-update

Description

Searches for the record which matches the query condition and calls **sieb-struct-set-field** to update the values.

Signature

```
(sieb-struct-update <sieb-obj-node> <query-cond>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the repeating business component node.
query-cond	Vector	Query condition.

Returns

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

Throws

None.

Location

sieb-struct-update.monk

sieb-struct-update2

Description

Searches for the record which matches the query condition in the association business components matching the specified business component handle and calls **sieb-struct-set-field** to update the values.

Signature

```
(sieb-struct-update2 <bus-com-hdl> <sieb-obj-node> <query-cond>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
sieb-obj-node	Path	Path to the repeating business component node.
query-cond	Vector	Query condition.

Returns

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

Throws

None.

Location

sieb-struct-update2.monk

sieb-struct-write

Description

Checks whether or not the records matching the key values exist. If they do, then the function conducts an update; otherwise, an insert. **Keys-rec** specifies a list of keys for searching. Key values are obtained for the business component path node.

Signature

```
(sieb-struct-write  <sieb-obj-node> <keys-vec>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the business component node (single instance).
keys-vec	Vector	Vector of Key Field names.

Returns

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

Throws

None.

Location

sieb-struct-write.monk

sieb-struct-write2

Description

Checks whether or not the records matching the key values exist in the associated business components matching the specified business component handle. If they do, then the function conducts an update; otherwise, an insert. **Keys-rec** specifies a list of keys for searching. Key values are obtained for the business component path node.

Signature

```
(sieb-struct-write2 <bus-com-hdl> <sieb-obj-node> <keys-vec>)
```

Parameters

Name	Type	Description
bus-com-hdl	Monk object	Business component handle.
sieb-obj-node	Path	Path to the business component node (single instance).
keys-vec	Vector	Vector of Key Field names.

Returns

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

Throws

None

Location

sieb-struct-write2.monk

sieb-struct-write-pick-mvg

Description

Allows you to insert or update a record and in addition specify values for both picklist and multi-value group fields for the record's business component. If the value of the **picklist-vec** or **mvg-vec** is **NULL** then that particular vector is ignored.

Signature

```
(sieb-struct-write-pick-mvg <sieb-obj-node> <bc-search-expr>
      <picklist-vec> <mvg-vec>)
```

Parameters

Name	Type	Description
sieb-obj-node	Path	Path to the business component node (single instance).
bc-search-expr	Vector	The search expression for the Business Component of the <sieb-obj-node> used to query for an existing record. If the query returns a record, the first record found is updated. If the query does not find a matching record then a new Siebel record is created. For accuracy this query should return at most one record.
picklist-vec	Vector	Picklist vector (see Additional Information, below)
mvg-vec	Vector	Multi-value group vector (see Additional Information, below)

Returns

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

Throws

None.

Location

sieb-struct-write-pick-mvg.monk

Additional Information

The Picklist and Multi-value group vectors have the following form:

```
(vector (vector <field-name> <search-expr>) (vector <field-name>
     <search-expr>) ... )
```

where:

Name	Type	Description
field-name	String	The name of the field on the BC that has either the pick list or multi value group.
search-expr	String	The search condition for the picklist or MVG business component. This condition should return one record only.

Examples

Search Expr:

```
(set! bc-search-expr (string-append "Id~=~" siebel-row-id "|"))
```

picklist-vec:

```
(set! picklist-vec (vector (vector "Unit of Measure" "Name~=~Each|")))
```

mvg-vec:

```
(set! mvg-vec (vector (vector "Product" (string-append "Id~=~" siebel-
     row-id "|"))))
```

7.3   Siebel General Functions

These Monk APIs are developed specifically to control basic communications between the Siebel Event-Driven e*Way and a Siebel application.

siebel-eventdriven-init

Description

Begins the initialization process for the e*Way. The function loads **stc_monkutils.dll** and any additional dynamic load libraries explicitly specified.

Signature

```
(siebel-eventdriven-init)
```

Parameters

None.

Returns

If successful, a Boolean true (#t); otherwise, a Boolean false (#f) and the e*Way shuts down.

Throws

None

Location

siebel-eventdriven-init.monk

siebel-eventdriven-startup

Description

A sample Monk function for e*Way startup.

Signature

```
(siebel-eventdriven-startup)
```

Parameters

None.

Return Values

If successful, the string **"SUCCESS"**; any other value indicates failure.

Throws

None.

Location

siebel-eventdriven.monk

siebel-eventdriven-connect

Description

Establishes a connection with the Siebel system, for Outgoing (e*Gate-to-Siebel) e*Ways only.

Signature

```
(siebel-eventdriven-connect)
```

Parameters

None.

Returns

If connection is established successfully, the string **"UP"**; any other value indicates failure to achieve a connection.

Throws

None.

Location

siebel-eventdriven-connect.monk

Additional Information

Incoming (Siebel-to-e*Gate) e*Ways use the function **siebel-eventdriven-dummy** in place of **siebel-eventdriven-connect**.

siebel-eventdriven-dummy

Description

Establishes a connection with the Siebel system, for Incoming (Siebel-to-e*Gate) e*Ways only.

Signature

```
(siebel-eventdriven-dummy)
```

Parameters

None.

Returns

An empty string ("") is provided for user modification.

Throws

None.

Location

siebel-eventdriven.monk

Additional Information

Outgoing (e*Gate-to-Siebel) e*Ways use the function **siebel-eventdriven-connect** in place of **siebel-eventdriven-dummy**.

siebel-eventdriven-verify-connect

Description

A sample Monk function for connection verification.

Signature

```
(siebel-eventdriven-verify-connect)
```

Parameters

None.

Returns

If connection is found to be live, returns the string **"UP"**; any other value indicates a live connection was not found.

Throws

None.

Location

siebel-eventdriven.monk

Notes

User should implement proper connection verification functions.

siebel-eventdriven-ack

Description

A sample Monk function for positive acknowledgement to the external system.

Signature

```
(siebel-eventdriven-ack)
```

Parameters

None.

Returns

An empty string ("") is provided for user modification.

Throws

None.

Location

siebel-eventdriven.monk

siebel-eventdriven-nack

Description

A sample Monk function for negative acknowledgement to the external system.

Signature

```
(siebel-eventdriven-nack)
```

Parameters

None.

Returns

An empty string ("") is provided for user modification.

Throws

None.

Location

siebel-eventdriven.monk

siebel-eventdriven-exchange

Description

A sample Monk function for message exchange.

Signature

```
(siebel-eventdriven-exchange)
```

Parameters

None.

Returns

A string is provided for user modification.

Throws

None.

Definition

```
(define siebel-eventdriven-exchange
 (lambda (   )
   (display "inside siebel-eventdriven-exchange - not used") (newline)
   ""
))
```

Location

siebel-eventdriven-exchange.monk

siebel-eventdriven-exchange-data

Description

A sample Monk function for data exchange.

Signature

```
(siebel-eventdriven-exchange-data)
```

Parameters

None.

Returns

An empty string (“”) is provided for user modification.

Throws

None.

Location

siebel-eventdriven.monk

siebel-eventdriven-return-empty-string

Description

A sample Monk function for returning an empty string.

Signature

```
(siebel-eventdriven-return-empty-string)
```

Parameters

None.

Return Values

An empty string (**""**).

Throws

None.

Location

siebel-eventdriven.monk

siebel-eventdriven-shutdown

Description

A sample Monk function for shutting down the e*Way.

Signature

```
(siebel-eventdriven-shutdown)
```

Parameters

None.

Returns

If successful, returns the string **"SUCCESS",** allowing an immediate shutdown to occur.

Throws

None.

Location

siebel-eventdriven.monk

siebel-debug-info

Description

Displays information resulting from any debug flags, using the Monk function **current-debug-port**.

Signature

```
(siebel-debug-info)
```

Parameters

None.

Returns

None.

Throws

None.

Location

siebel-eventdriven.monk

siebel-log-info

Description

Displays output information using the Monk function **current-output-port**.

Signature

```
(siebel-log-info)
```

Parameters

None.

Returns

None.

Throws

None.

Location

siebel-eventdriven.monk

7.4 Example Functions

The following Monk script and Monk functions have been defined for the Sample Schema included with the Siebel Event-Driven e*Way.

EnqueueSiebelReply

Description

Monk script that tests the first 11 characters of the specified message to see if they match **SIEBELERROR**. If the message starts with **SIEBELERROR**, the Monk function **iq-put** is called, and the error is placed on the IQ as an event of type **QueuedError.** Valid replies are published to the IQ as the default Event Type for this collaboration, **QueuedReply.**

Signature

```
(EnqueueSiebelReply <message-string>)
```

Parameters

Name	Type	Description
message-string	String	The message string returned by Siebel.

Returns

None.

Throws

None.

Location

EnqueueSiebelReply.monk

See also

Siebel to e*Gate: Query-Reply Example on page 47

Siebel to e*Gate: COM Server Example on page 53

siebel-com-account-exchange

Description

Retrieves internally-tagged data fields from Siebel. It first creates a **sieb-account** structure internally, then passes it to the Siebel COM Object Manager

Signature

```
(siebel-com-account-exchange)
```

Parameters

None.

Returns

A string:

- Upon success, the reply from **sieb-struct-lookup** is mapped to a string and returned.

- If an error is detected, a message string having **SIEBELERROR** as the first 11 characters is returned.

Throws

None.

Location

siebel-com-account-exchange.monk

See also

siebel-com-account-query

Description

Queries the Siebel application according to the specified query. The query is then stored in a global Monk variable (**siebel-current-query**) and the scheduler is invoked.

Signature

```
(siebel-com-account-query <message-string>)
```

Parameters

Name	Type	Description
message-string	String	The message string forming the basis for the query.

Returns

An empty string ("").

Throws

None.

Location

siebel-com-account-query.monk

See also

Siebel to e*Gate: Query-Reply Example on page 47

Siebel to e*Gate: COM Server Example on page 53

siebel-eventdriven-account-post

Description

Attempts to post the specified message to Siebel; upon failure, returns the message.

Signature

```
(siebel-eventdriven-account-post <message-string>)
```

Parameters

Name	Type	Description
message-string	String	The message string to be posted.

Returns

Upon failure, the original message.

Throws

None.

Location

siebel-eventdriven-account-post.monk

See also

e*Gate to Siebel Example on page 41

7.5 Generic e*Way Functions

The functions described in this section are implemented in the e*Way Kernel layer and control the e*Way's most basic operations. They can be used only 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. These functions are located in **stcewgenericmonk.exe**.

The current set of basic Monk functions is:

event-commit-to-egate on page 134

event-rollback-to-egate on page 135

event-send-to-egate on page 135

event-send-to-egate-ignore-shutdown on page 136

event-send-to-egate-no-commit on page 136

get-logical-name on page 137

insert-exchange-data-event on page 137

send-external-up on page 138

send-external-down on page 138

shutdown-request on page 139

start-schedule on page 139

stop-schedule on page 140

waiting-to-shutdown on page 140

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 136

event-send-to-egate-no-commit on page 136

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 135

event-send-to-egate-no-commit on page 136

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 134

event-rollback-to-egate on page 135

event-send-to-egate on page 135

event-send-to-egate-ignore-shutdown on page 136

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 Enterprise Manager).

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 89

Zero Wait Between Successful Exchanges on page 90

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.

Index

U

Up Timeout parameter **89**
User Interface Objects Layer (Siebel) **13**
User name **63**

W

waiting-to-shutdown function **140**
writing conventions **9**

Z

Zero Wait Between Successful Exchanges parameter **90**