SeeBeyond™ eBusiness Integration Suite

# e*Way Intelligent Adapter for Siebel EIM User's Guide

*Release 4.5.2*

SEEBEYOND™

Version 20020308044425.

# 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, Windows 2000, or UNIX systems
- Windows-style GUI operations
- Siebel 99 or 2000, and Siebel Enterprise Integration Manager (EIM)

## 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 EIM is frequently referred to as the Siebel EIM 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.

Chapter 1

# Introduction

## 1.1 Overview

The e*Way Intelligent Adapter for Siebel EIM provides connectivity between e*Gate and Siebel 99 Front Office using the Enterprise Integration Manager (EIM). It provides an inbound and outbound batch interface option to or from another system, through e*Gate.

The e*Way is the central command center for the entire interface process. It provides a development environment that allows automatic Event Type Definition of Siebel's interface tables to provide easy, drag-and-drop Collaborations between Siebel and virtually any other application. The Siebel EIM e*Way is designed to generate all **.ifb** file definitions, providing the ability to control the entire interface from one source.

Also provided is an execution environment that oversees Siebel's EIM processes and dynamically creates Siebel EIM configuration files. It incorporates detailed error logs, bad-Event journalling, and reprocessing capabilities for the failed imported records during operation of the Siebel Enterprise Integration Manager. All Application Logic and Business Rules are enforced using Siebel EIM.

An auxiliary file-handling e*Way is included to assist in the feeding and logging of data processed by e*Gate. This e*Way is useful for error-handling purposes and as a substitute for a source or target e*Way for testing purposes.

### 1.1.1 System Features

#### Interface Monitoring

The Siebel EIM provides error messages, which are stored in a flat file on the Application Server. This file can be read via the Siebel Administration Screen.

Siebel-specific monitoring capabilities are custom log files created during processing. All errors encountered during EIM processing are captured in a log file to be viewed, fixed and reprocessed.

### System Alerts

Typically a third party product is used to monitor the Process Identification Numbers (PID) on the Siebel Server to notify administrators if the system or process goes down. The e*Gate Monitor tracks the flow of PIDs, which are below the NT service level, to ensure that crucial process are running properly.

### Transactional Alerts

Transactional alerts pertain to errors that occur during the processing of any customized pieces on both the Siebel and e*Gate side. In Siebel this pertains to errors which occur during the failure of any SQL statements, looping through internal tables and other logic errors during the EIM export. In e*Gate, transactional errors pertain to any errors which may occur during the business rules configured in e*Gate.

If an error occurs in either the Siebel export, or in e*Gate, the e*Gate Monitor can be configured to display any error Events.

### Auditing

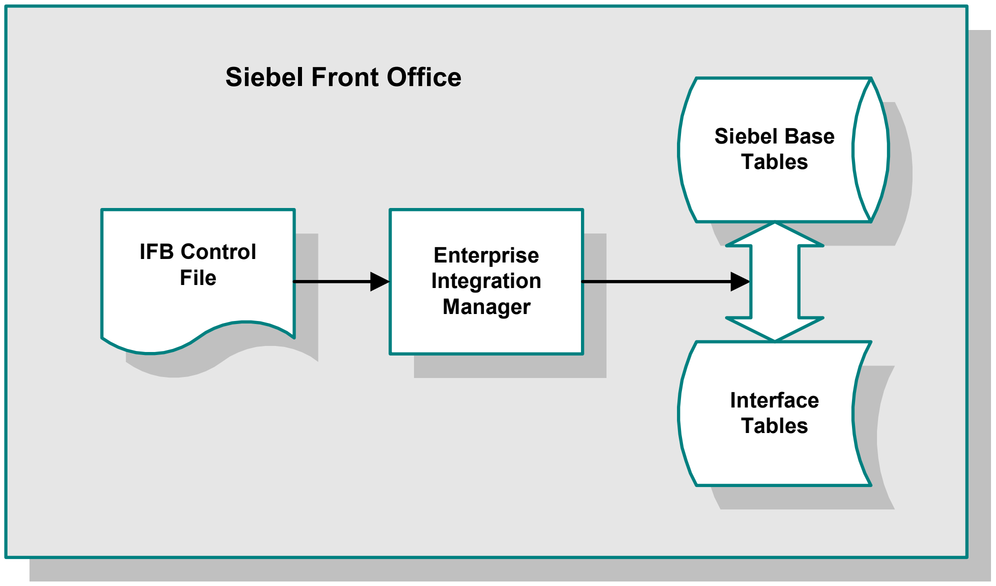
Error counts and totals are returned to the e*Way through variables. When a non-zero return code is given, these errors are written to the standard e*Way error logs.

## 1.2 Communicating with Siebel

### 1.2.1 Siebel EIM

The Siebel Enterprise Integration Manager is the first step in exporting, and the last step in importing, data. The database consists of two main groups of tables, interface tables and base tables. The Siebel client application communicates directly with the highly normalized base tables. The interface tables are used as a staging area for importing, exporting, deleting and merging logical groups of data. Each interface table represents a subset of the data in a specific base table.

**Figure 1** Siebel Enterprise Integration Manager



When run, EIM coordinates the transfer of data between the base and interface tables. In addition, EIM creates and writes codes and row IDs to the base tables that directly correspond to the complex set of business rules used by the Siebel client application to display data. The EIM process must run on the server also running the Transaction Processor.

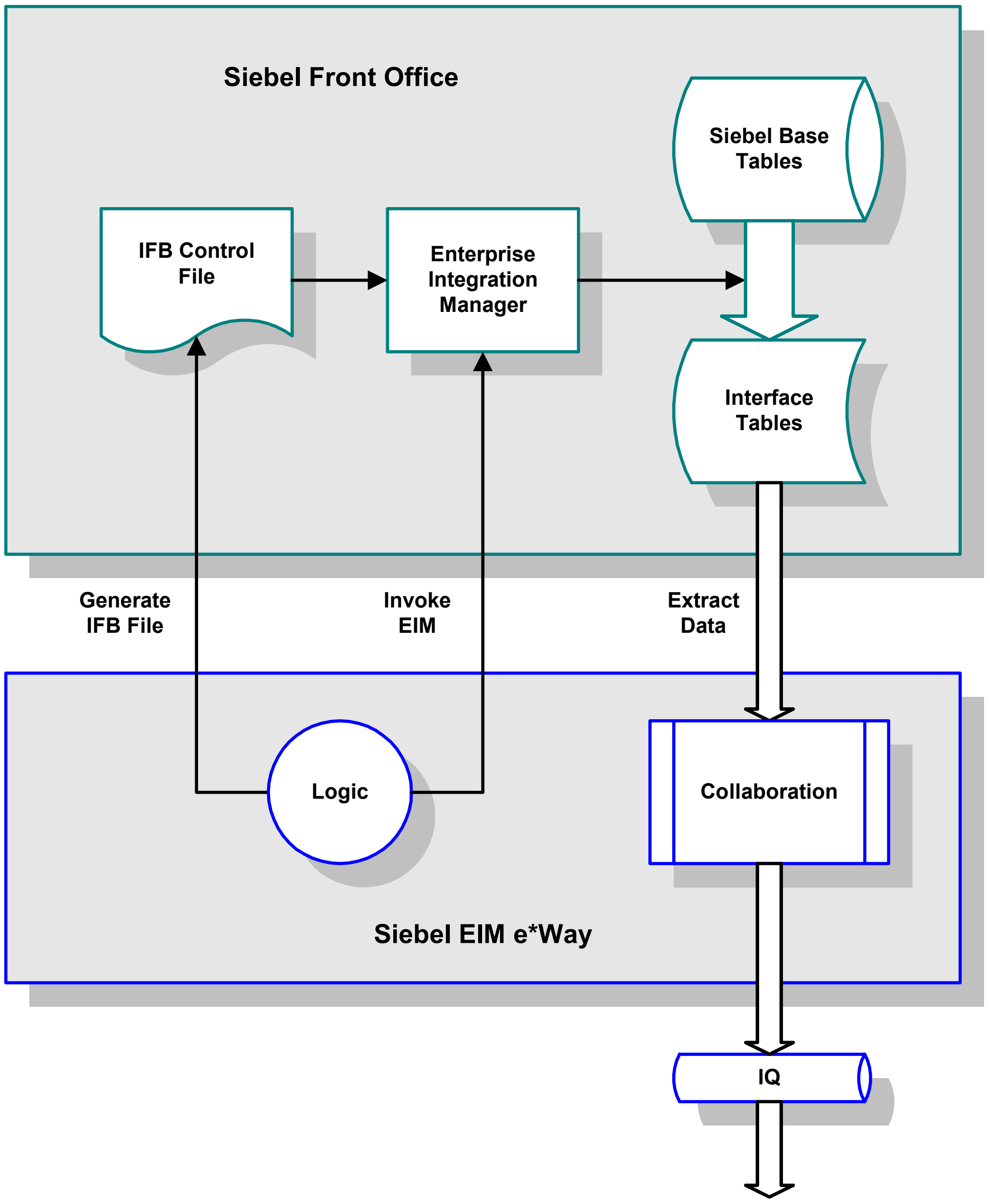
### 1.2.2 IFB Control File

A control file (***.ifb**) is used to determine what data types are loaded and how. The control file follows a certain format—it tells the interface how to log into the database and what process to run. It also lists the columns that the interface does *not* populate (in Siebel-inbound mode), thus preventing erroneous error messages. The IFB control file can be generated automatically by the e*Way or created manually, as configured by the user.

## 1.3 The Siebel EIM e*Way

### 1.3.1 Siebel to e*Gate

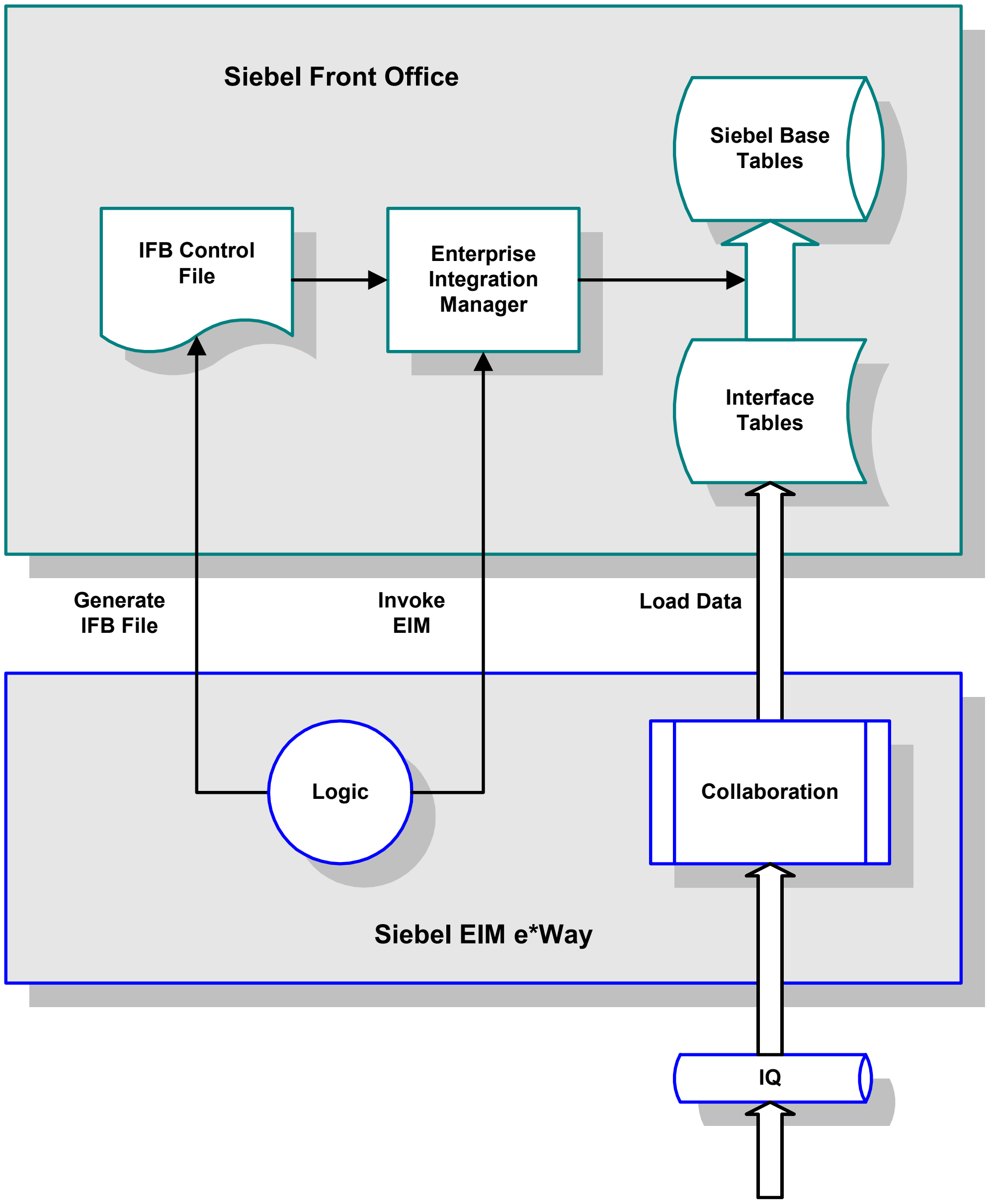
**Figure 2** Siebel-to-e*Gate Process Flow



Following a prescribed schedule, the Siebel EIM e*Way sends an IFB file and invokes the Siebel EIM process. Following the definitions in the IFB control file, the EIM copies data from the Siebel Base Tables into the Interface Tables. The e*Way extracts the data from the Interface Tables and maps it into the desired Event Type Definition. The data then is passed to an Intelligent Queue for subsequent processing and/or routing to the target application by other e*Gate components.

### 1.3.2 e*Gate to Siebel

**Figure 3** e*Gate-to-Siebel Process Flow



Following a prescribed schedule, the Siebel EIM e*Way extracts information from an e*Gate Intelligent Queue. The e*Way generates an IFB control file and sends it to Siebel. It then inserts validated rows of data into the interface table. Records that do not successfully load into the interface table are written to an error file for reprocessing. After the interface tables are populated, the e*Way initiates the Siebel EIM to load the data from the Interface Tables to the Siebel Base Tables.

### 1.3.3 e*Way Components

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

- An executable file, **stcewgenericmonk.exe**, installed as part ot e*Gate Integrator
- An accompanying dynamic load library, **stc_monksiebeleim.dll**, which extends the executable file to form the Siebel EIM e*Way
- An ancillary e*Way executable, **stcewfile.exe**, for error-file handling—refer to the *Standard e*Way Intelligent Adapters User's Guide* for information on this e*Way.
- A default configuration file, **SiebelEim.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**.

Chapter 2

# Installation

This chapter describes the requirements and procedures for installing the e*Way Intelligent Adapter for Siebel EIM.

## 2.1 System Requirements

To use the e*Way Intelligent Adapter for Siebel EIM, you need the following for e*Way executable, configuration, library, and script files:

1. An e*Gate Participating Host, version 4.5.1 or later.
2. A TCP/IP network connection.
3. An appropriate database e*Way (ODBC, Oracle, or Sybase).
4. Sufficient free disk space to accommodate e*Way files:
    - Approximately 45.9 KB on Windows systems
    - Approximately 82.4 KB on Solaris systems
    - Approximately 81.1 KB s on AIX systems

**Note:** *Additional disk space is required to process and queue the data that this e*Way processes; the amount necessary varies, based on the type and size of the data being processed, and any external applications performing the processing.*

### 2.1.1 Supported Operating Systems

**Note:** *The e*Gate Enterprise Manager GUI runs only on the Windows operating system.*

The e*Way Intelligent Adapter for Siebel EIM is available on the following operating systems:

- Windows 2000, Windows 2000 SP 1, and Windows 2000 SP 2
- Windows NT 4.0 SP 6a
- Solaris 2.6, Solaris 7, and Solaris 8
- AIX 4.3.3

**Japanese**

- Windows 2000, Windows 2000 SP 1, and Windows 2000 SP 2
- Windows NT 4.0 SP 6a

***Note:*** *If you are running AIX 4.3.3, you should ensure that you have* **bos.rte.libpthreads** *version 4.3.3.51 or later. Earlier versions contain a recognized bug that can produce a memory leak. A patch is available from IBM.*

### 2.1.2 External System Requirements

The e*Way Intelligent Adapter for Siebel EIM supports the following applications:

- Siebel Server version 5.0 or 6.0
- Siebel Front Office 99.5, 99.6, and 2000

**Japanese**

- Siebel Server version 5.0 or 6.0
- Siebel Front Office 2000

***Note:*** *If you are using a non-English version of Siebel, you need to modify the Monk function* **siebel-eim-run-eim.monk**. *See* **Siebel IFB Functions** *on page 103.*

### 2.1.3 External Configuration Requirements

Most installations of Siebel applications require some customization of the Data Model to meet the client’s specific requirements. We assume that any customization of Siebel required for your implementation has been performed previous to the installation of e*Gate. No special configuration of the Siebel application is required to interface with e*Gate.

## 2.2 Installing the e*Way

**Note:** *It is not necessary to install the e*Gate components on the Siebel Application server; however, the e*Way must have access to the Siebel File system.*

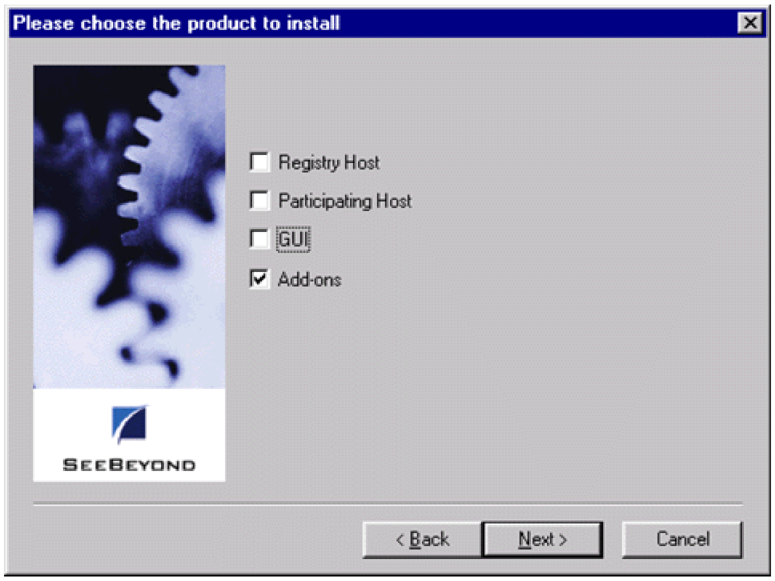
### 2.2.1 Windows Systems

#### Installation Procedure

**Note:** *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond. You must have Administrator privileges to install this e*Way.*

1. Log in as an Administrator on the workstation on which you want to install the 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. If the CD-ROM drive's Autorun feature is enabled, the setup application should launch automatically. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
5. The InstallShield setup application launches. Follow the on-screen instructions until you come to the **Choose Product** screen.

**Figure 4** Choose Product Dialog

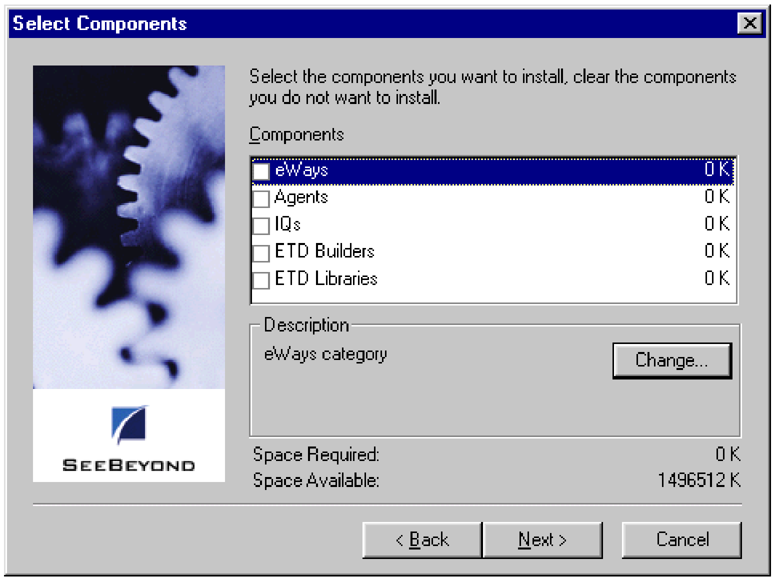

Please choose the product to install

- [ ] Registry Host
- [ ] Participating Host
- [ ] GUI
- [✓] Add-ons

6 Check **Add-ons**, then click **Next**. Again follow the on-screen instructions.

7 When the **Select Components** dialog box appears, highlight—but do not check—**eWays** and then click **Change**.

**Figure 5** Select Components Dialog (1)



Select Components

Select the components you want to install, clear the components you do not want to install.

Components

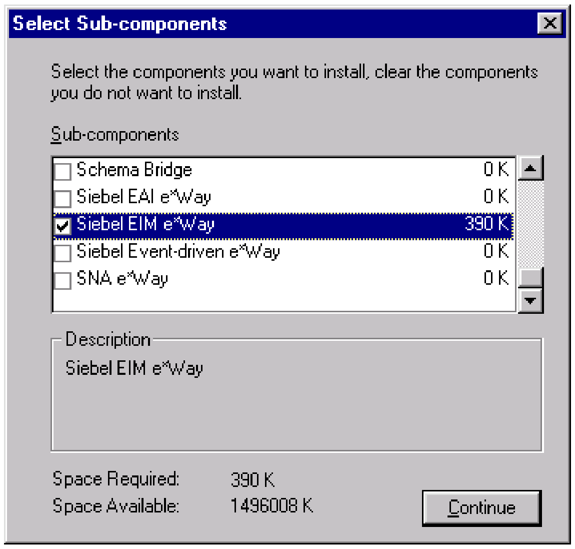
| | |
|---|---|
| eWays | 0 K |
| Agents | 0 K |
| IQs | 0 K |
| ETD Builders | 0 K |
| ETD Libraries | 0 K |

Description
eWays category

Space Required: 0 K
Space Available: 1496512 K

8 When the **Select Sub-components** dialog box appears, check the **Siebel EIM e*Way**.

**Figure 6** Select e*Way Dialog



Select Sub-components

Select the components you want to install, clear the components you do not want to install.

Sub-components

| | |
|---|---|
| Schema Bridge | 0 K |
| Siebel EAI e*Way | 0 K |
| ✓ Siebel EIM e*Way | 390 K |
| Siebel Event-driven e*Way | 0 K |
| SNA e*Way | 0 K |

Description
Siebel EIM e*Way

Space Required: 390 K
Space Available: 1496008 K

9 Click **Continue**, and the **Select Components** dialog box reappears.

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

## 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 1** Participating Host & Registry Host

| Subdirectories | Files |
|---|---|
| \bin\ | stc_monksiebeleim.dll |
| \configs\stcewgenericmonk\ | SiebelEim.def<br>siebelEim3.6To4.1Rule.txt |
| \monk_library\ | ewsiebeleim.gui |
| \monk_library\ewsiebeleim\ | CaptureProcessOutput.monk<br>popen-layer.monk<br>Siebel_EIM_server_tasks.ssc<br>Siebel_EIM_server_tasks_2000.ssc<br>Siebel_IFB.ssc<br>siebel-eim-create-ifb.monk<br>siebel-eim-init.monk<br>siebel-eim-run-eim.monk<br>siebel-eim-server-capture.monk<br>siebel-eim-utils.monk<br>siebel-eim.monk |

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

**Table 2** Registry Host Only

| Subdirectories | Files |
|---|---|
| \ | stcewsiebeleim.ctl |

## Environment Configuration

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

### 2.2.2 UNIX Systems

#### Installation Procedure

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

1. Log onto the workstation containing the CD-ROM drive and, if necessary, mount the drive.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. At the shell prompt, type

```
cd /cdrom
```

4. Start the installation script by typing:

```
setup.sh
```

5. A menu appears, containing several options. Select the **Install e*Way** option, and follow any additional on-screen directions.

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

#### Subdirectories and Files

The preceding installation procedure 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 3** Participating Host & Registry Host

| Subdirectories | Files |
|---|---|
| /bin/ | stc_monkpipes.dll |
| /configs/stcewgenericmonk/ | SiebelEim.def<br>siebelEim3.6To4.1Rule.txt |
| /monk_library/ | ewsiebeleim.gui |
| /monk_library/ewsiebeleim/ | Siebel_EIM_server_tasks.ssc<br>Siebel_IFB.ssc<br>siebel-eim.monk<br>siebel-eim-create-ifb.monk<br>siebel-eim-init.monk<br>siebel-eim-run-eim.monk<br>siebel-eim-server-capture.monk<br>siebel-eim-utils.monk |

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

**Table 4** Registry Host Only

| Subdirectories | Files |
|---|---|
| / | stcewsiebeleim.ctl |

## Environment Configuration

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

## 2.3 Optional Example Files

The installation CD-ROM contains sample schema, **Siebel_EIM_Post** and **Siebel_EIM_Extract**, located in the **samples\ewsiebeleim** directory. To use a schema, you must load it onto your system using the following procedure. See **Sample Schema** on page 39 for descriptions of the sample schema and instructions regarding its use.

Two versions of each schema are provided:

- **Siebel_EIM_Post** (Siebel-inbound e*Way example)
  - Use **Siebel_EIM_Post.zip** for Siebel 99
  - Use **Siebel_EIM_Post_2000.zip** for Siebel 2000
- **Siebel_EIM_Extract** (Siebel-outbound e*Way example)
  - Use **Siebel_EIM_Extract.zip** for Siebel 99
  - Use **Siebel_EIM_Extract_2000.zip** for Siebel 2000

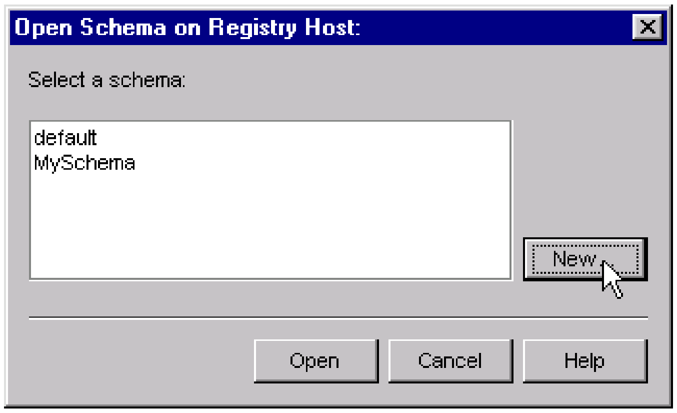
***Note:*** *The Siebel EIM e*Way must be properly installed on your system before you can run the sample schema.*

### 2.3.1 Installation Procedure

**To load a sample schema:**

1 Invoke the **Open Schema** dialog box and select **New** (see Figure 7).

**Figure 7** Open Schema Dialog


Open Schema on Registry Host:

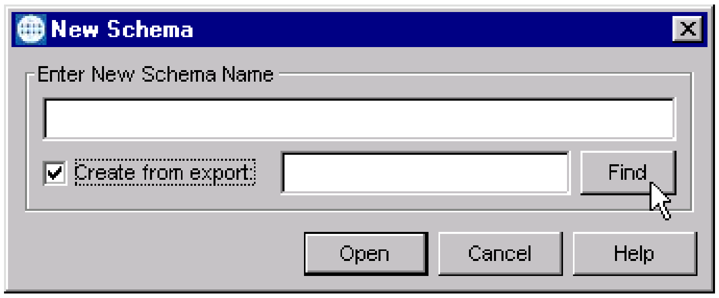
Select a schema:

default
MySchema

2 Type the name you want to give to the schema (for example, **Post.Sample**)

3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 8).

**Figure 8** 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.*

## 2.3.2 Subdirectories and Files

The preceding process creates the following subdirectories and installs the following files within the **\eGate\client\eGateImport** tree on the Participating Host, and commits them to their own schemas on the Registry Host.

**Table 5** Subdirectories and Files - Siebel EIM Extract

| Subdirectories | Files |
|---|---|
| \ | Siebel_EIM_Extract.ctl<br>Siebel_EIM_Extract.exp |
| \Siebel_EIM_Extract\runtime\configs\stcewfile\ | Siebel_EIM_Eater.cfg<br>Siebel_EIM_Eater.sc |
| \Siebel_EIM_Extract\runtime\configs\stcewgenericmonk\ | Siebel_EIM_Extract.cfg<br>Siebel_EIM_Extract.sc |
| \Siebel_EIM_Extract\runtime\monk_scripts\common\ | export_accounts.dsc<br>export_cleanup.dsc<br>export_ifb_gen.tsc<br>from_s_account_if.dsc<br>Monk_Function_Ret.ssc<br>sap_data_contract.ssc<br>Siebel_accounts.ssc<br>Siebel_IFB.ssc |

**Table 6** Subdirectories and Files - Siebel EIM Extract 2000

| Subdirectories | Files |
|---|---|
| \ | Siebel_EIM_Extract_2000.ctl<br>Siebel_EIM_Extract_2000.exp |
| \Siebel_EIM_Extract_2000\runtime\configs\stcewfile\ | Siebel_EIM_Eater.cfg<br>Siebel_EIM_Eater.sc |
| \Siebel_EIM_Extract_2000\runtime\configs\stcewgeneric monk\ | SiebelEim_Extract.cfg<br>SiebelEim_Extract.sc |
| \Siebel_EIM_Extract_2000\runtime\monk_scripts\commo n\ | export_accounts.dsc<br>export_cleanup.dsc<br>export_ifb_gen.tsc<br>from_s_account_if.dsc<br>Monk_Function_Ret.ssc<br>sap_data_contract.ssc<br>Siebel_accounts.ssc<br>Siebel_IFB.ssc |

**Table 7** Subdirectories and Files - Siebel EIM Post

| Subdirectories | Files |
|---|---|
| \ | Siebel_EIM_Post.ctl<br>Siebel_EIM_Post.exp |
| \Siebel_EIM_Post\runtime\configs\stcewfile\ | Siebel_EIM_Eater_Error.cfg<br>Siebel_EIM_Eater_Error.sc<br>Siebel_EIM_Feeder.cfg<br>Siebel_EIM_Feeder.sc<br>Siebel_EIM_Feeder_Error.cfg<br>Siebel_EIM_Feeder_Error.sc |
| \Siebel_EIM_Post\runtime\configs\stcewgenericmonk\ | Siebel_EIM_Post.cfg<br>Siebel_EIM_Post.sc<br>Siebel_EIM_Post_Error.cfg<br>Siebel_EIM_Post_Error.sc |
| \Siebel_EIM_Post\runtime\data\ | sap_account_10.dat |
| \Siebel_EIM_Post\runtime\monk_scripts\common\ | import_accounts.dsc<br>import_accounts_error.dsc<br>import_cleanup.dsc<br>import_ifb_gen.tsc<br>sap_data_contract.ssc<br>Siebel_accounts.ssc<br>Siebel_accounts_batch.ssc<br>Siebel_IFB.ssc<br>to_s_account_if.tsc<br>to_s_account_if_error.tsc |

**Table 8** Subdirectories and Files - Siebel EIM Post 2000

| Subdirectories | Files |
|---|---|
| \ | Siebel_EIM_Post_2000.ctl<br>Siebel_EIM_Post_2000.exp |
| \Siebel_EIM_Post_2000\runtime\configs\stcewfile\ | Siebel_EIM_Eater_Error.cfg<br>Siebel_EIM_Eater_Error.sc<br>Siebel_EIM_Feeder.cfg<br>Siebel_EIM_Feeder.sc<br>Siebel_EIM_Feeder_Error.cfg<br>Siebel_EIM_Feeder_Error.sc |
| \Siebel_EIM_Post_2000\runtime\configs\stcewgenericmonk\ | Siebel_EIM_Post.cfg<br>Siebel_EIM_Post.sc<br>Siebel_EIM_Post_Error.cfg<br>Siebel_EIM_Post_Error.sc<br>SiebelEim.def<br>SiebelEim_Post.cfg<br>SiebelEim_Post.sc |
| \Siebel_EIM_Post_2000\runtime\data\ | sap_account_10.dat |
| \Siebel_EIM_Post_2000\runtime\monk_scripts\common\ | import_accounts.dsc<br>import_accounts_error.dsc<br>import_cleanup.dsc<br>import_ifb_gen.tsc<br>sap_data_contract.ssc<br>Siebel_accounts.ssc<br>Siebel_accounts_batch.ssc<br>Siebel_IFB.ssc<br>to_s_account_if.tsc<br>to_s_account_if_error.tsc |

Chapter 3

# System Implementation

This chapter describes the procedure for creating a functional Siebel-e*Gate system incorporating the Siebel EIM e*Way. Refer to the *e*Gate Integrator User's Guide* for additional information.

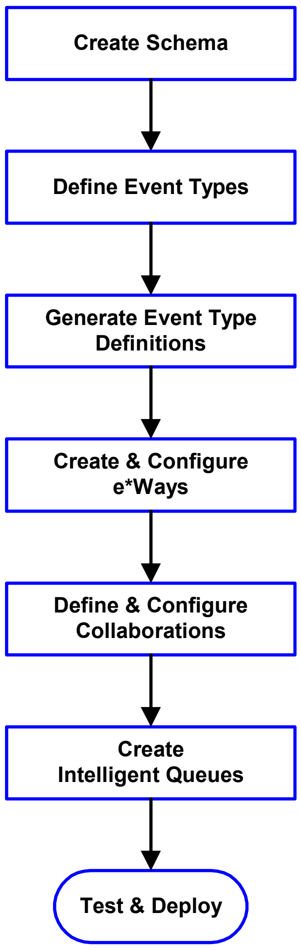
## 3.1 Overview

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:

### 3.1.1 Implementation Sequence

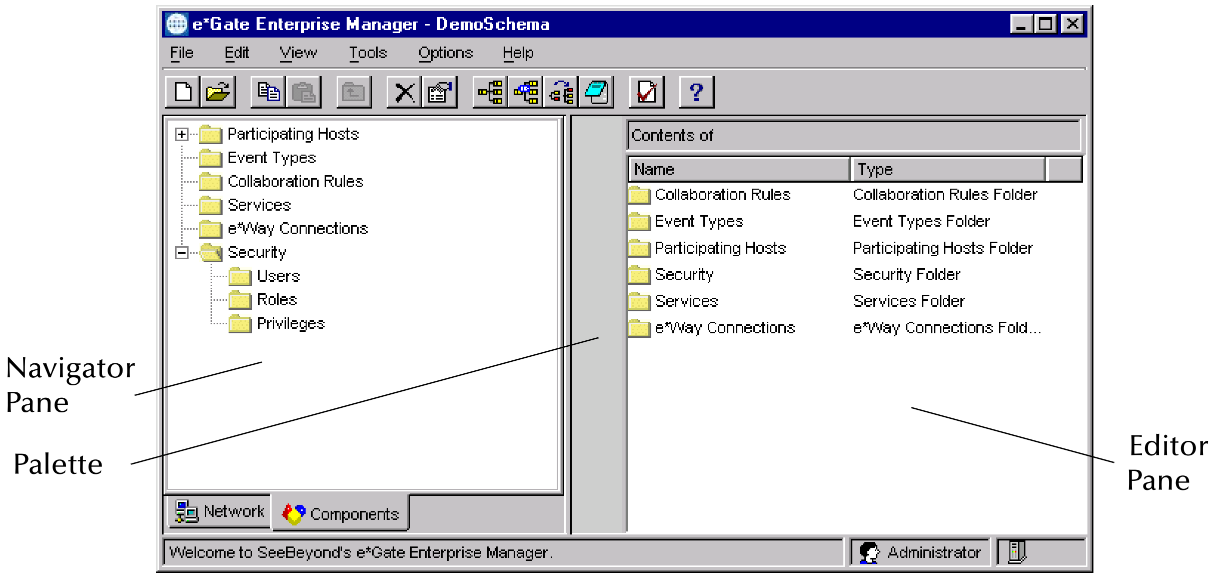

Create Schema
Define Event Types
Generate Event Type Definitions
Create & Configure e*Ways
Define & Configure Collaborations
Create Intelligent Queues
Test & Deploy

1. The first step is to create a new Schema—the subsequent steps apply only to this Schema (see **Creating a Schema** on page 30).
2. The second step is to define the Event Types you are transporting and processing within the Schema (see **Creating Event Types** on page 31).
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 **Creating Event Type Definitions** on page 32).
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 36).
6. Now you need to create Intelligent Queues to hold published Events (see **Creating Intelligent Queues** on page 37
7. Finally, you must test your Schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

## 3.2 The e*Gate Enterprise Manager

First, here is a brief look at the e*Gate Enterprise Manager. The general features of the e*Gate Enterprise Manager window are shown in Figure 9. See the *e*Gate Integrator User's Guide* for a detailed description of the features and use of the Enterprise Manager.

**Figure 9** e*Gate Enterprise Manager Window (Components View)



Contents of

| Name | Type |
|---|---|
| Collaboration Rules | Collaboration Rules Folder |
| Event Types | Event Types Folder |
| Participating Hosts | Participating Hosts Folder |
| Security | Security Folder |
| Services | Services Folder |
| e*Way Connections | e*Way Connections Fold... |

Welcome to SeeBeyond's e*Gate Enterprise Manager.

Use the Navigator and Editor panes to view the 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.

## 3.3 Creating a Schema

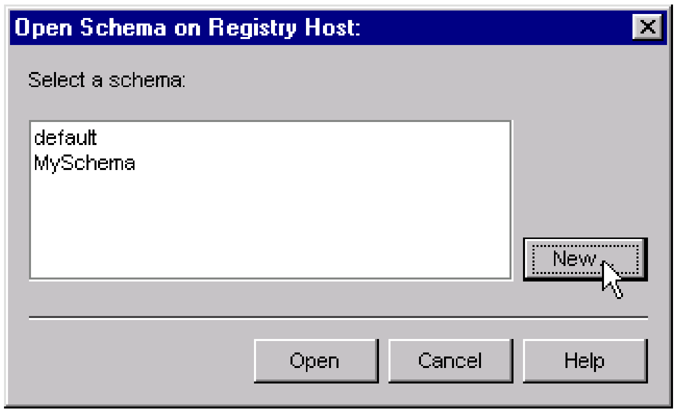
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

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

### To select or create a schema

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

**Figure 10** Open Schema Dialog



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

## 3.4 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**.

## 3.5 Creating Event Type Definitions

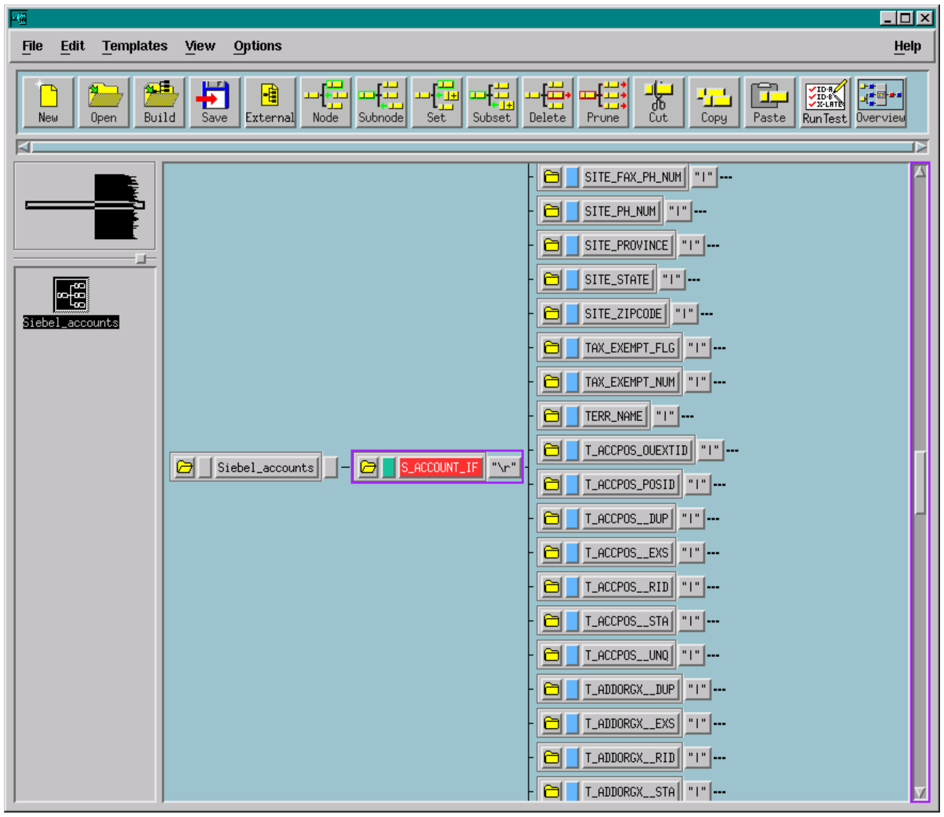
Before e*Gate can process any data to or from a Siebel system, you must create an Event Type Definition to package and route that data within the e*Gate system. See the *e*Gate Integrator User's Guide* for additional information about Event Type Definitions and the e*Gate ETD Editor.

### 3.5.1 Obtaining the Siebel Data Structure

Once the functional needs of the interface have been defined, the input data must be mapped between e*Gate and the interface tables. To accomplish this, custom structure metadata must first be imported from Siebel into e*Gate. This process consists of importing the Interface Table data structure from Siebel into the e*Way using the e*Gate Event Type Definition Builder.

The e*Gate ETD Builder automatically creates an Event Type Definition in e*Gate that is used to create the mapping for the interface. The ETD Builder reads the data definition of from the Siebel Interface Tables and defines the column attributes, which are displayed in a graphical representation. An illustration of an interface table, S_ACCOUNT_IF, is depicted in Figure 11.

**Figure 11** Interface Table Structure Example



## 3.5.2 Using the ETD Editor's Build Tool

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

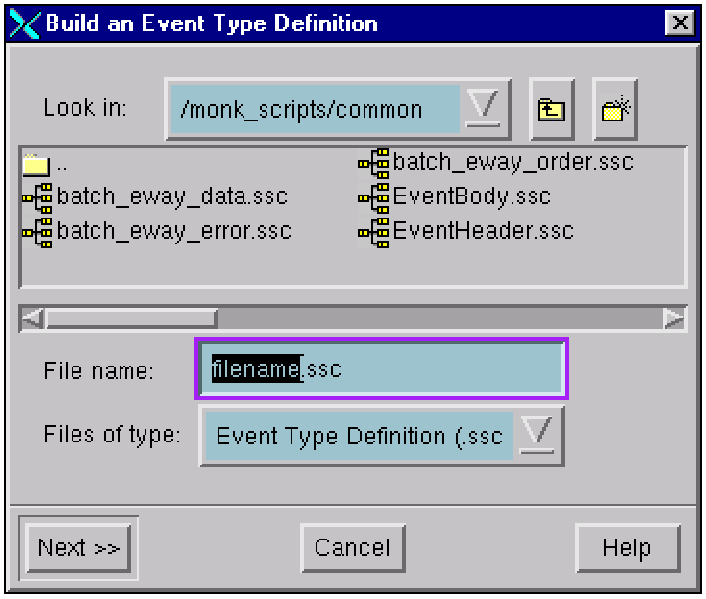
**Note:** *Be sure to set the Default Editor to* **Monk**, *from the* **Options** *menu in the e*Gate Enterprise Manager.*

### To create an Event Type Definition using the Build tool

1. Launch the ETD Editor by clicking 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 12** Build Event Type Definition Dialog



Build an Event Type Definition

Look in: /monk_scripts/common

..
batch_eway_data.ssc
batch_eway_error.ssc
batch_eway_order.ssc
EventBody.ssc
EventHeader.ssc

File name: filename.ssc

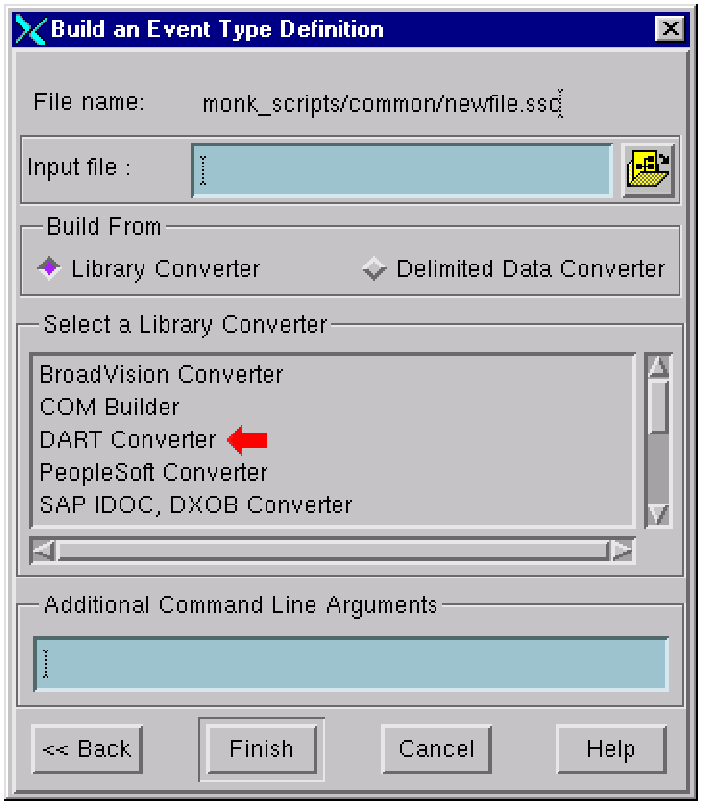
Files of type: Event Type Definition (.ssc

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 13.

**Figure 13** Building the ETD



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

6 Under *Select a Library Converter*, select **DART Converter**.

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

8 Click **Finish**.

9 The DART Converter Wizard automatically builds the ETD file.

## 3.5.3 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**

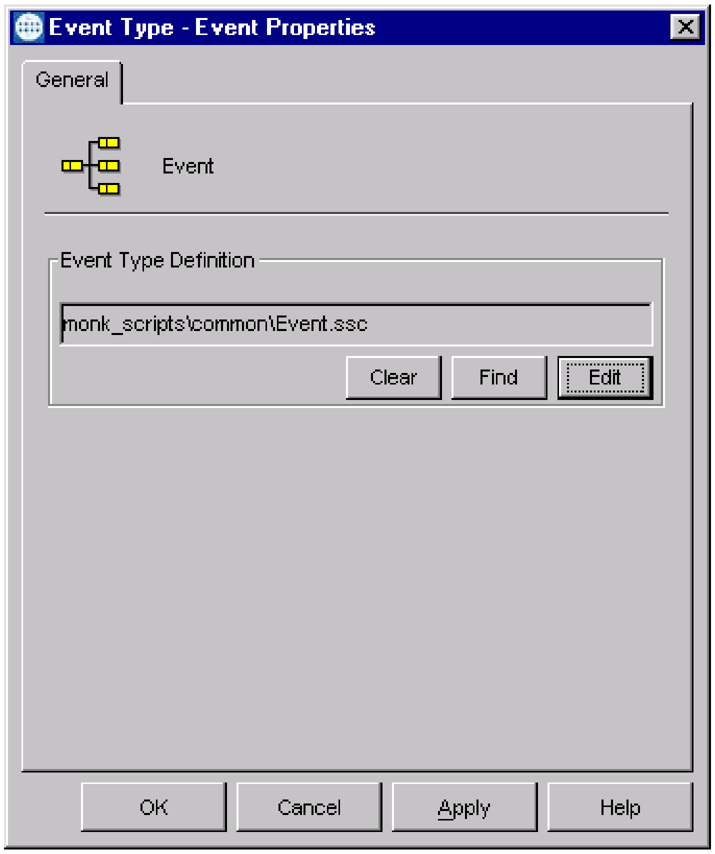
1 In the Enterprise Manager window, select the **Event Types** folder in the Navigator/Components pane.

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 in the toolbar).

The Event Type Properties dialog box appears (see Figure 14).

**Figure 14** Event Type Properties Dialog Box



monk_scripts\common\Event.ssc

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.

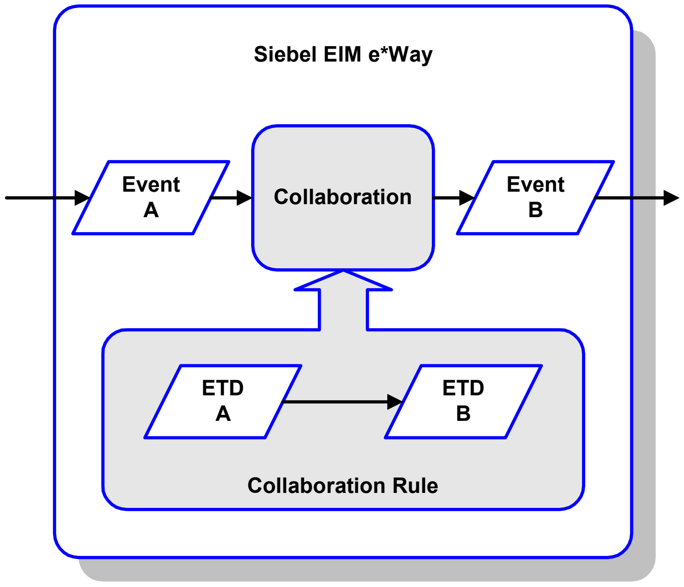
Each Event Type is now associated with the specified Event Type Definition.

## 3.6 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 15** Collaborations



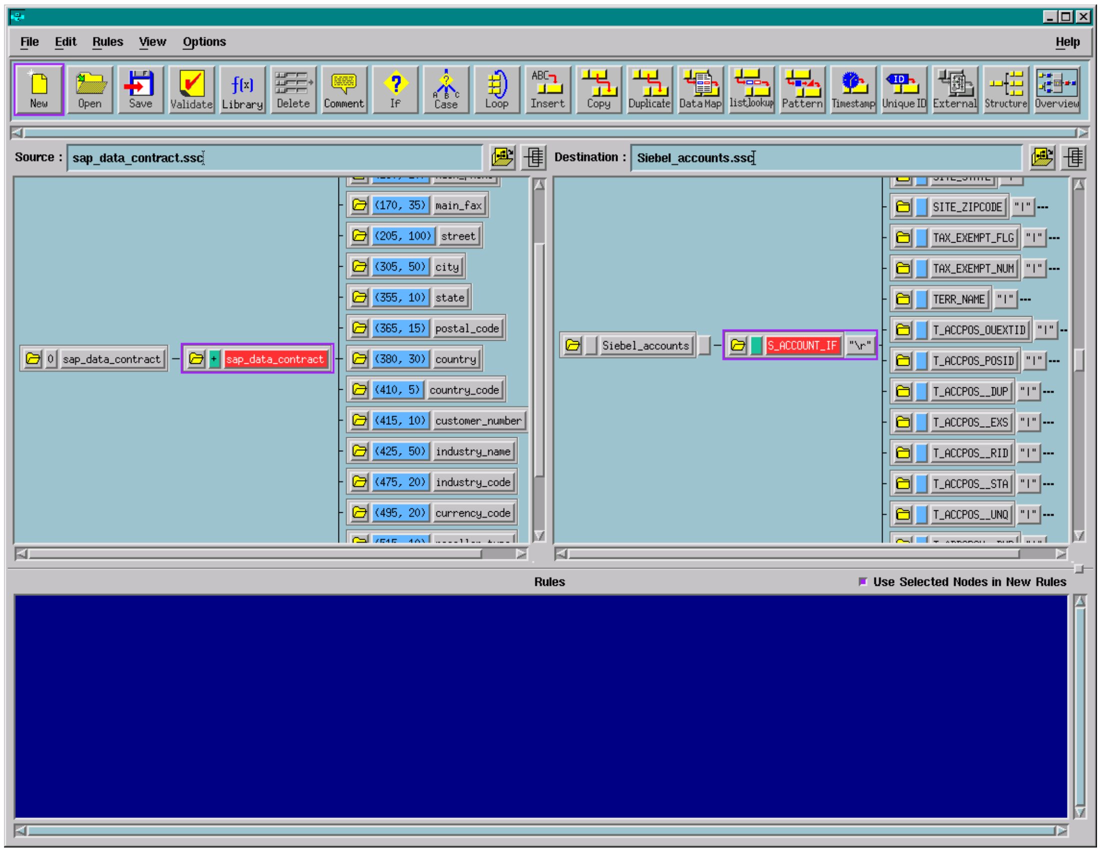
The Collaboration is driven by a Collaboration Rule, which defines the relationship between the incoming and outgoing ETDs. You can use an existing Collaboration Rule, or use the Monk programming language to write a new Collaboration Rule script. Once you have written and successfully tested a script, 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 EIM e*Way can be found in **Sample Schema** on page 39.

### 3.6.1 Using the Collaboration Rules Editor

**Figure 16** The Collaboration Rules Editor



Source : sap_data_contract.ssc

Destination : Siebel_accounts.ssc

Figure 16 illustrates a Collaboration that maps an incoming flat file to the **S_ACCOUNT_IF** interface table, another flat file. This is the simplest method of moving data into a Siebel interface table. The source and destination ETDs are shown in the left and right panes, respectively.

## 3.7 Creating Intelligent Queues

The final step is to create and associate an IQ for the Siebel EIM 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.

## 3.8 Auditing

Error counts and totals are returned to the e*Way through variables. When a non-zero return code is given, these errors are written to the standard e*Way error logs.

### 3.8.1 Error logging—Inbound to Siebel

Errors are logged in the Siebel interface tables, a custom file is created by the Siebel EIM e*Way, and the standard error logs are supplied by the e*Gate components. The Siebel EIM e*Way monitors the status of EIM by capturing the task state for the initiated interface. The state changes to **Completed**, thus signaling the end of the EIM process.

The Siebel EIM e*Way interrogates the status of the **IF_ROW_STAT** column in the interface table where the batch number equals the records just loaded. The Siebel EIM e*Way extracts all records from the interface tables where the **IF_ROW_STAT** does not equal **Imported**, and sends an alert to the Administrator that manual intervention is required.

***Note:*** *If you are using a non-English version of Siebel, you need to modify the Monk function* **siebel-eim-run-eim.monk**. *See* **Siebel IFB Functions** *on page 103.*

### 3.8.2 Error logging—Outbound from Siebel

Although Siebel lacks an alert notification facility to notify administrators, it does log the success or failure of each row during the export process. If the exported data cannot be extracted from the database, a message is recorded in the e*Way log and a Notification is sent to the e*Gate Monitor.

### 3.8.3 Journaling

Errors encountered during normal e*Gate processing are addressed by normal functionality. Each Logical Unit of Work (LUW) that encounters errors during translation is placed into a journal for later viewing, correction and reprocessing.

## 3.9 Known Issues and Limitations

When using **s_account_if** with EIM you may run into this error:

```
"Warning: Bounded picklist value S_ORG_EXT.CUST_STAT_CD failed 11".
```

This issue has been logged with Siebel Support and can be found by searching Siebel for the topic *Changing IFMGR use of Bounded PickList Values*. We recommend contacting your Siebel Technical Account Manager to resolve this issue on your system.

Below are several alternative workarounds that can be implemented by those technically well-versed with Siebel.

1. Add an extension column to the interface table and use this column to directly import the picklist data to the base table.
2. Turn off bounded picklist behavior on the interface table.
3. Import the values into the List Of Values table before importing the data using EIM. Following is an example query that can be used to populate the **S_LST_OF_VAL_IF** table with Account Status data:

```
Insert S_LST_OF_VAL_IF
(IF_ROW_STAT, ROW_ID, IF_ROW_BATCH_NUM, TYPE, VAL, NAME, LAND_CD)
select distinct 'X', rownum, 1, 'cust_stat_code', version,
   version, 'ENU'
FROM S_PROD_INT_IF
where IF_ROW_BATCH_NUM = 2;
```

4. Use the SeeBeyond Siebel EIM e*Way.
5. Insert only those columns needed into the EIM interface table. For example,

```
ONLY_BASE_COLUMNS=S_ORG_EXT.NAME,S_ORG_EXT.LOC,
                  S_ORG_EXT.MAIN_FAX_PH_NUM,
                  S_ORG_EXT.MAIN_PH_NUM
```

## 3.10 Sample Schema

Sample implementations are available in the **\samples\ewsiebeleim\** directory of the e*Gate CD-ROM.

- **Siebel_EIM_Extract** - Siebel-outbound schema example
- **Siebel_EIM_Post** - Siebel-inbound schema example

These samples can be used to test your system following installation and, if appropriate, as templates you can modify to produce your own schema.

See **Optional Example Files** on page 23 for installation instructions.

### 3.10.1 Siebel_EIM_Extract

The Siebel-to-e*Gate example, **Siebel_EIM_Extract**, sets up a single instance of the Siebel EIM e*Way and also of the File e*Way, having the logical names shown in the following table.

| e*Way Type | Logical Name |
|---|---|
| Siebel EIM e*Way | Siebel_EIM_Extract |
| File e*Way | Siebel_EIM_Eater |

It also sets up an Intelligent Queue, with the logical name **Siebel_EIM_Extract_IQ**. The schema contains one main collaboration, **Account Extract**, and several sub-collaborations, all of which are described on following pages.

There is one data type, **Siebel_EIM_Accts**, representing incoming data from a Siebel system. This data type is passed from one component to another following two Publisher/Subscriber actions, as outlined below and diagrammed in Figure 17.

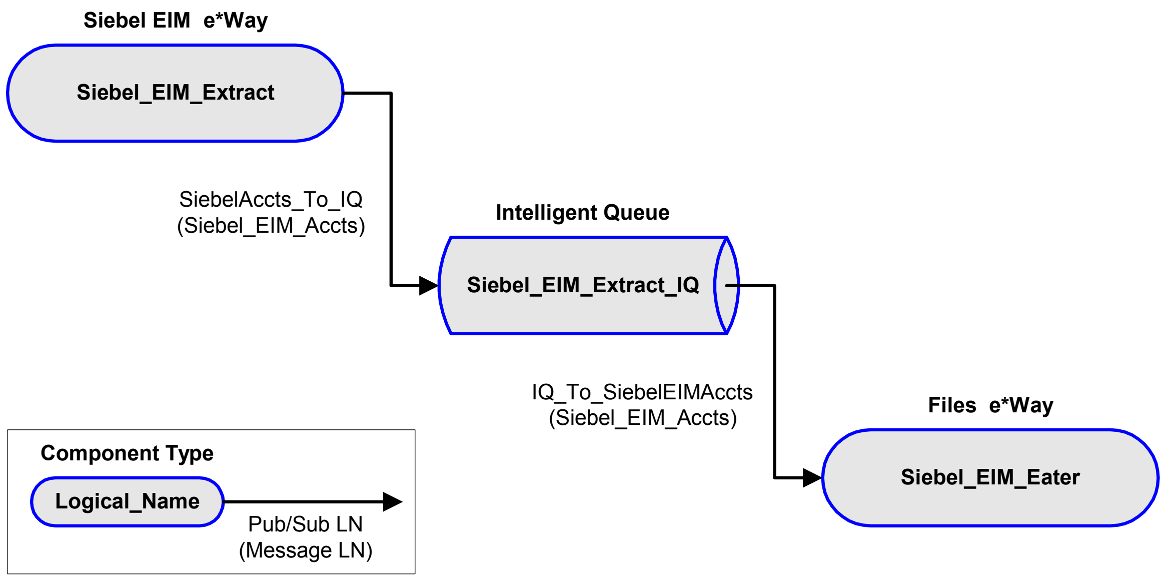
#### SiebelAccts_To_IQ

This publisher sends the Siebel account data from **Siebel_EIM_Extract** to the IQ.

#### IQ_To_SiebelAccts

This subscriber sends the Siebel account data from the IQ to **Siebel_EIM_Eater**.

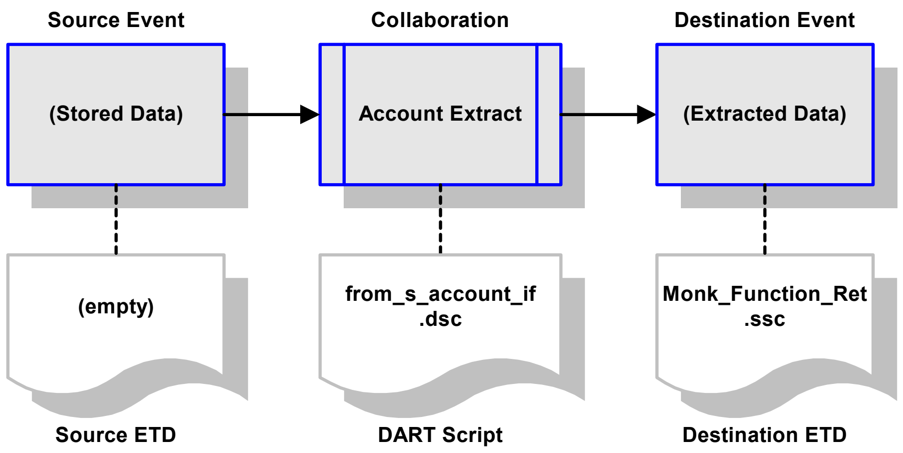
**Figure 17** Siebel_EIM_Extract Schema



## Account Extract

This Collaboration, based on the Monk script **from_s_account_if.dsc**, provides an example for the Siebel EIM e*Way running in the Outbound-from-Siebel mode. The script is invoked by the e*Way to extract and process Events from Siebel and place them into an Intelligent Queue.

**Figure 18** Account Extract Collaboration

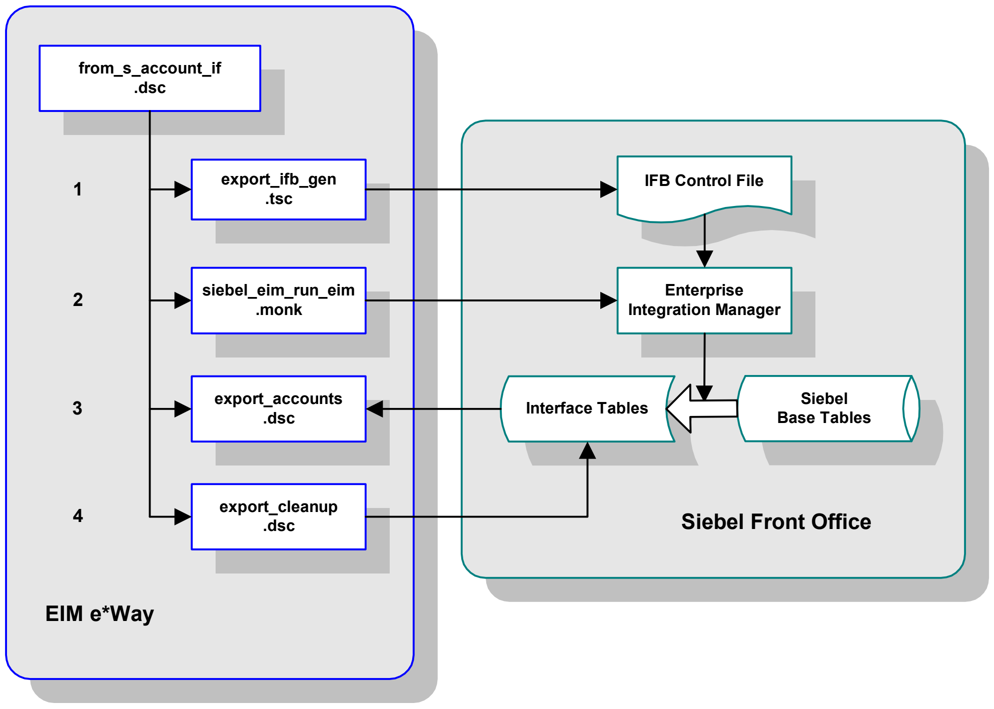


The source ETD is undefined, since this a DART-mode Collaboration, and assumes the structure of data received from the Siebel system. The destination ETD, **Monk_Function_Ret.ssc**, represents the structure desired for subsequent data processing.

In response to this Collaboration, the Siebel EIM e*Way functions as shown in Figure 19. In this scenario, the e*Way is instantiated as **Siebel_EIM_Extract**. After initially checking the Siebel State File, the e*Way creates and sends the IFB Control File **export_ifb_gen.tsc** to Siebel. It then invokes the EIM using the Monk function **siebel_eim_run_eim**, which transfers the data from the Siebel Base Tables to the Interface Tables.

Once the tables have been loaded by the EIM, the e*Way extracts the data from the Siebel Interface Tables following the DART script **export_accounts.dsc**. Using the DART script **export_cleanup.dsc**, the data is then deleted from the Interface Tables.

**Figure 19** Account Extract Functionality



### 3.10.2 Siebel_EIM_Post

The e*Gate-to-Siebel example, **Siebel_EIM_Post**, sets up two instances of the Siebel EIM e*Way and three instances of the File e*Way, having the logical names shown in the following table.

| e*Way Type | Logical Name |
|---|---|
| Siebel EIM e*Way | Siebel_EIM_Post |
| | Siebel_EIM_Post _Error |
| File e*Way | Siebel_EIM_Eater_Error |
| | Siebel_EIM_Feeder |
| | Siebel_EIM_Feeder_Error |

It also sets up an Intelligent Queue, with the logical name **Siebel_EIM_Post_IQ**. The schema contains two main collaborations, Account Post and Account Post (Corrected Error), and several sub-collaborations, all of which are described on following pages.

There are two data types, **SAPDataContract** and **SiebelAcctsError**, respectively representing incoming data from an SAP application, and erroneous data retrieved from Siebel. These data types are passed from one component to another following several Publisher/Subscriber actions, as outlined below and diagrammed in Figure 20.

**SAPAccts_To_IQ**

This publisher sends the SAP account data from **Siebel_EIM_Feeder** to the IQ.

**IQ_To_SAPAccts**

This subscriber sends the SAP account data from the IQ to **Siebel_EIM_Post**.

**SiebelAcctsError_To_IQ**

This publisher returns any errors from **Siebel_EIM_Post** to the IQ.

**IQ_To_SiebelAcctsErrorFile**

This subscriber sends the errors from the IQ to **Siebel_EIM_Eater_Error**.

**SiebelAcctsErrorFile_To_IQ**

This publisher sends the corrected data from **Siebel_EIM_Feeder_Error** to the IQ.

**IQ_To_SiebelAcctsError**

This subscriber sends the corrected data from the IQ to **Siebel_EIM_Post_Error**.

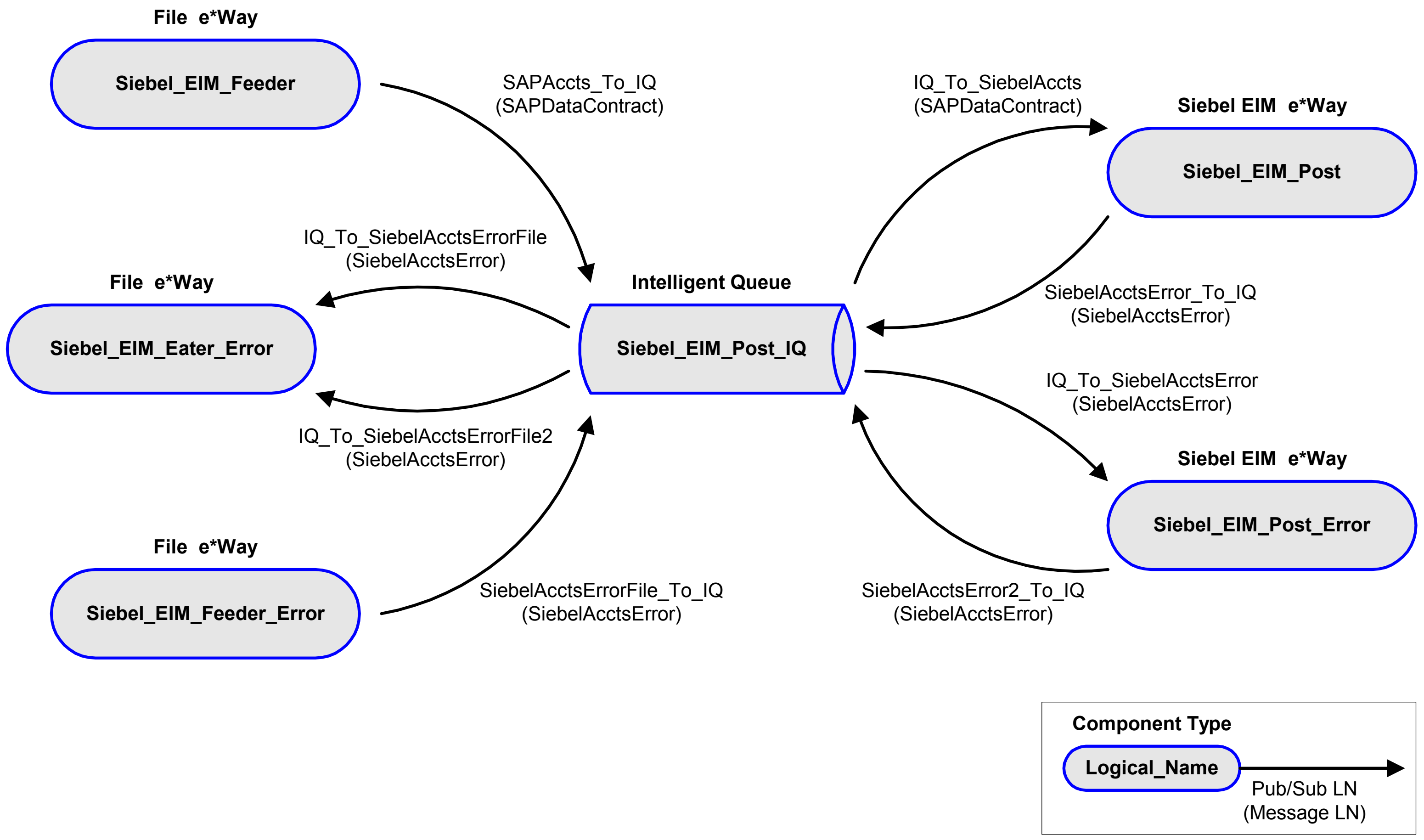
**SiebelAcctsError2_To_IQ**

This publisher sends any new errors from **Siebel_EIM_Post_Error** to the IQ.

**IQ_To_SiebelAcctsErrorFile2**

This subscriber sends the new errors from the IQ to **Siebel_EIM_Eater_Error**.

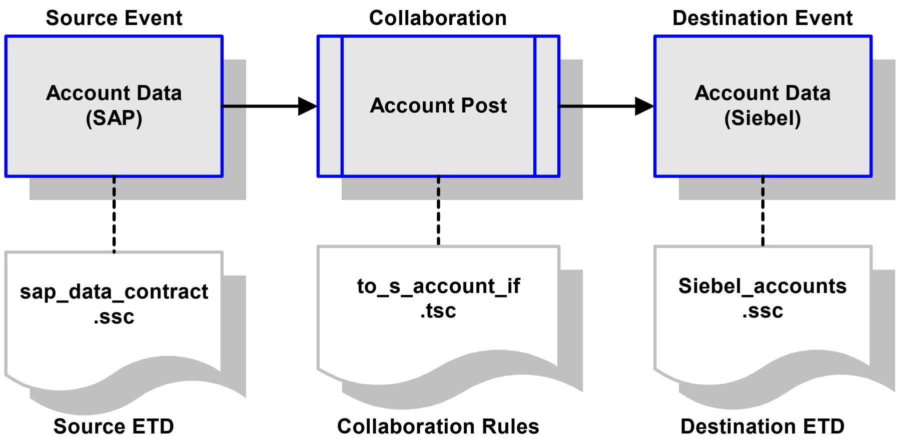
**Figure 20** Siebel_EIM_Post Schema



## Account Post

This Collaboration, based on the Monk script **to_s_account_if.tsc**, provides an example of the Siebel EIM e*Way running in the Inbound-to-Siebel mode. When the Siebel EIM e*Way receives an Event from e*Gate, via an Intelligent Queue, the script is invoked by the e*Way to process the Event. In this example, the input data from e*Gate is interpreted as an Event containing the required data field values for a customer account.

**Figure 21** Account Post Collaboration

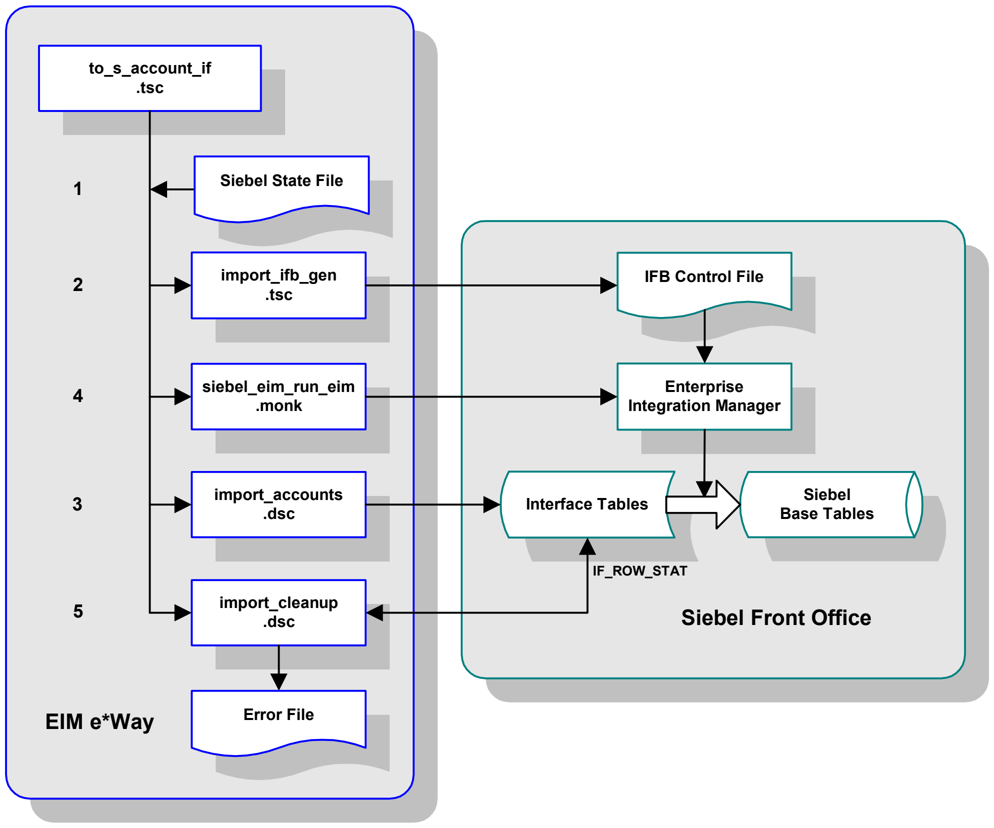


The source ETD, **sap_data_contract.ssc** represents the structure of data received from, or sent to, an SAP R/3 system. The destination ETD, **Siebel_accounts.ssc**, represents the structure of data required by the Siebel system.

In response to this Collaboration, the Siebel EIM e*Way functions as shown in Figure 22. In this scenario, the e*Way is instantiated as **Siebel_EIM_Post**. After initially checking the Siebel State File, the e*Way creates and sends the IFB Control File **import_ifb_gen.tsc** to Siebel. It then loads the incoming data into the Siebel Interface Tables following the DART script **import_accounts.dsc**.

Once the tables have been loaded the e*Way invokes the EIM using the Monk function **siebel_eim_run_eim**, which transfers the data from the Interface Tables to the Siebel Base Tables. It then checks the **IF_ROW_STAT** column in the Interface Tables using the DART script **import_cleanup.dsc**. Data that has not been transferred successfully from the Interface Tables to the Base Tables is retrieved and written to an error file. Everything else is simply deleted from the Interface Tables.

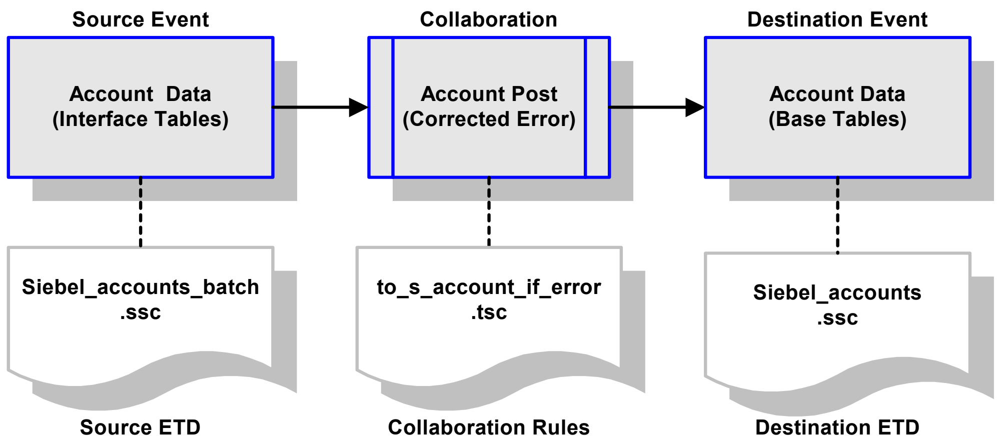
**Figure 22** Account Post Functionality



## Account Post (Corrected Error)

The Collaboration script **to_s_account_if_error.tsc** works in the same manner as **to_s_account_if.tsc**, with the exception that it accepts input data formatted according to the Siebel Interface Table structure. The difference is most apparent in the **Database Interface Function**.

**Figure 23** Account Post (Corrected Error) Collaboration

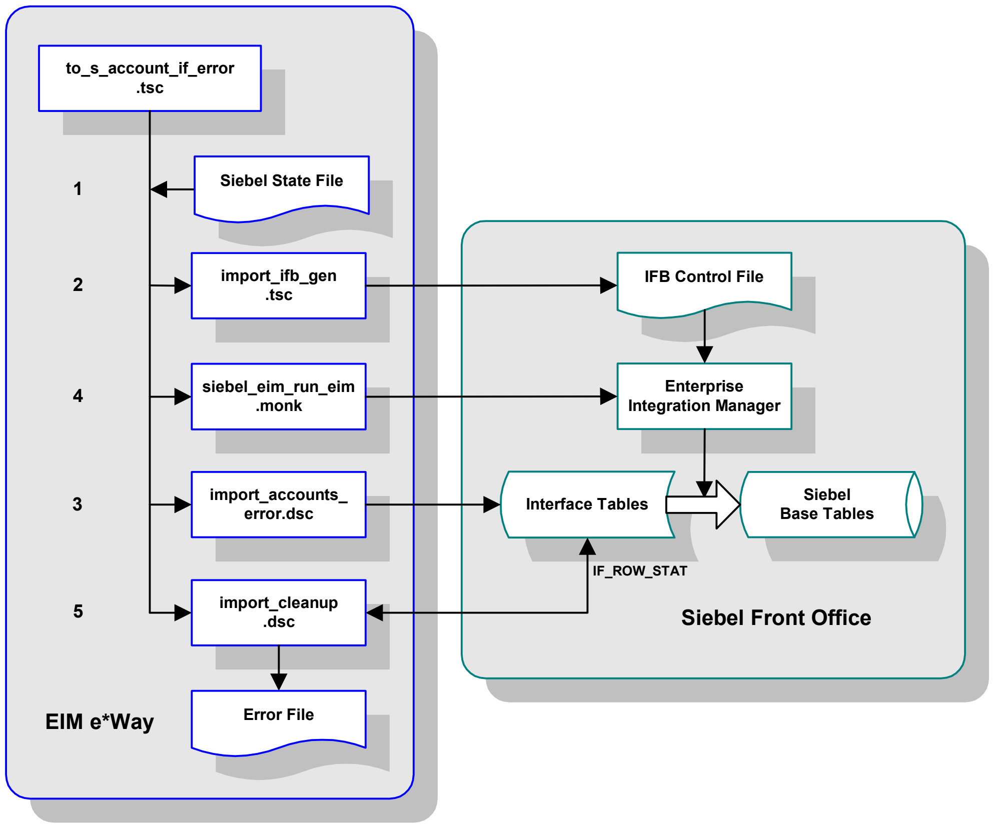

Source Event
Collaboration
Destination Event
Account Data (Interface Tables)
Account Post (Corrected Error)
Account Data (Base Tables)
Siebel_accounts_batch .ssc
to_s_account_if_error .tsc
Siebel_accounts .ssc
Source ETD
Collaboration Rules
Destination ETD

The source ETD, **Siebel_accounts_batch.ssc** represents the structure of data as it exists in the Siebel Interface Tables. As before, the destination ETD, **Siebel_accounts.ssc**, also represents the structure of data required by the Siebel system.

In response to this Collaboration, the Siebel EIM e*Way functions as shown in Figure 24. In this scenario, the e*Way is instantiated as **Siebel_EIM_Post_Error**. After initially checking the Siebel State File, the e*Way creates and sends the IFB Control File **import_ifb_gen.tsc** to Siebel. It then loads the incoming data into the Siebel Interface Tables following the DART script **import_accounts_error.dsc**.

Once the tables have been loaded the e*Way invokes the EIM using the Monk function **siebel_eim_run_eim**, which transfers the data from the Interface Tables to the Siebel Base Tables. It then checks the **IF_ROW_STAT** column in the Interface Tables using the DART script **import_cleanup.dsc**. Data that still has not been transferred successfully from the Interface Tables to the Base Tables is again retrieved and written to an error file. Successfully transferred records are simply deleted from the Interface Tables.

**Figure 24** Account Post (Corrected Error) Functionality



Chapter 4

# Setup Procedures

This chapter describes the procedures for customizing the SeeBeyond e*Way Intelligent Adapter for Siebel EIM to operate within your production system.

## 4.1 Overview

After installing the Siebel EIM e*Way, you must set it up to work with your system. 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:

## 4.2 Setting Up the e*Way

**Note:** *The e*Gate Enterprise Manager GUI runs only on the Windows operating system.*

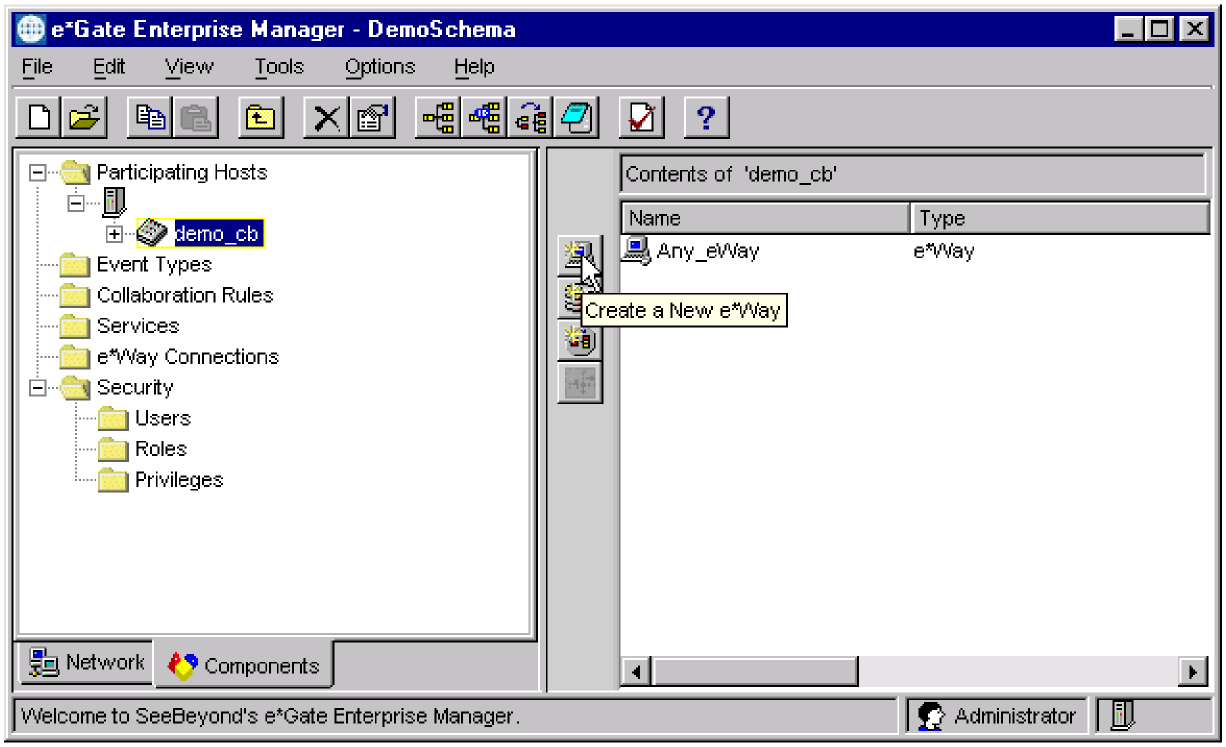
### 4.2.1 Creating the e*Way

The first step in implementing an e*Way is to define the e*Way component using the e*Gate 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 25** 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.

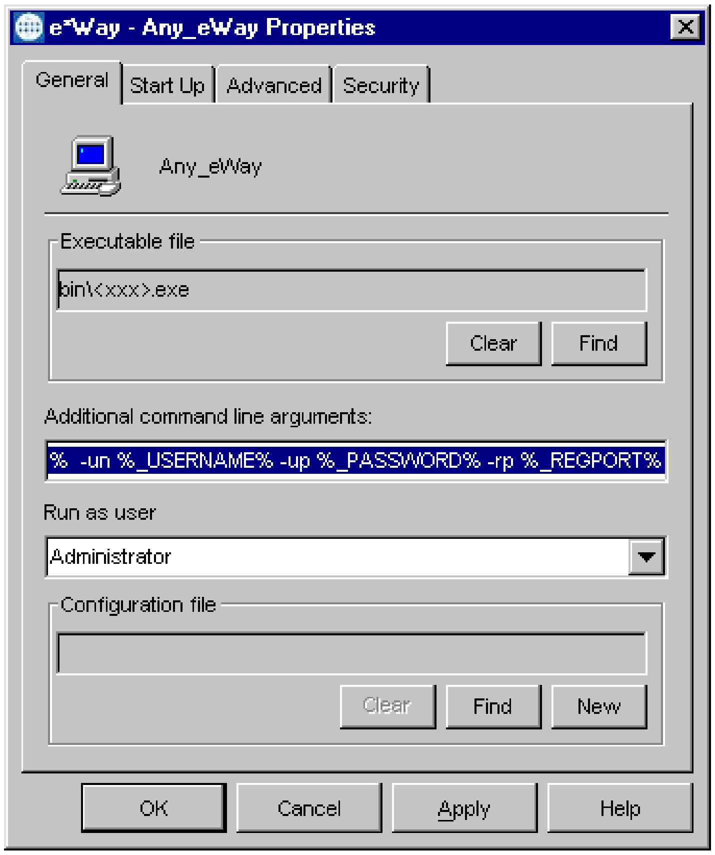
## 4.2.2 Modifying e*Way Properties

**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 26).

**Note:** *The executable and default configuration files used by this e*Way are listed in* **e*Way Components** *on page 15.*

**Figure 26** e*Way Properties (General Tab)



Any_eWay

Executable file: bin\<xxx>.exe

Additional command line arguments: % -un %_USERNAME% -up %_PASSWORD% -rp %_REGPORT%

Run as user: Administrator

Configuration file:

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

## 4.2.3 Configuring the e*Way

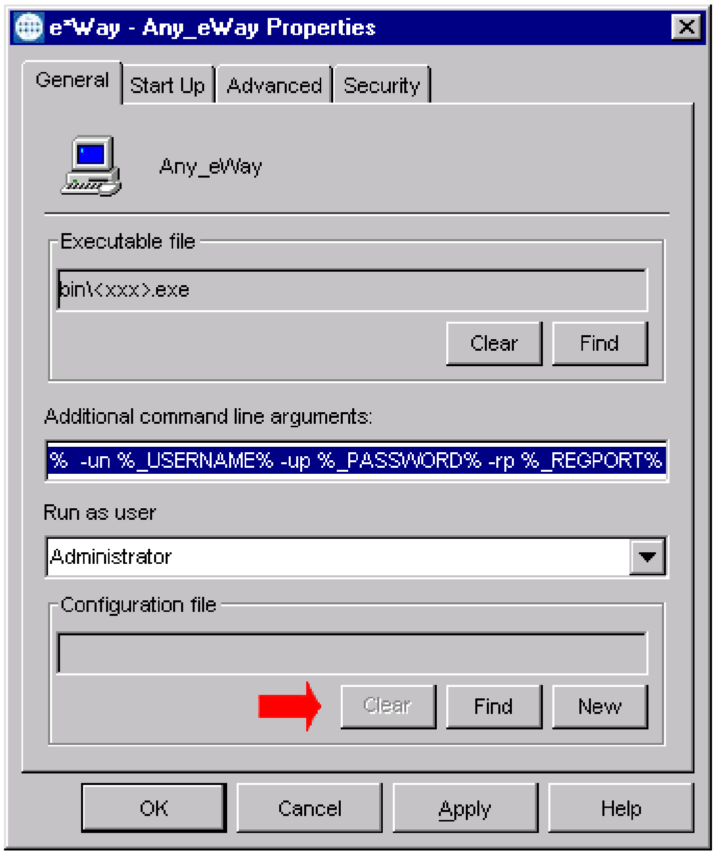
The e*Way's default configuration parameters are stored in an ASCII text file with a **.def** extension. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (**.cfg**) file.

### 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 15.*

**Figure 27** e*Way Properties - General Tab



e*Way - Any_eWay Properties

General | Start Up | Advanced | Security

Any_eWay

Executable file: bin\<xxx>.exe

Additional command line arguments: % -un %_USERNAME% -up %_PASSWORD% -rp %_REGPORT%

Run as user: Administrator

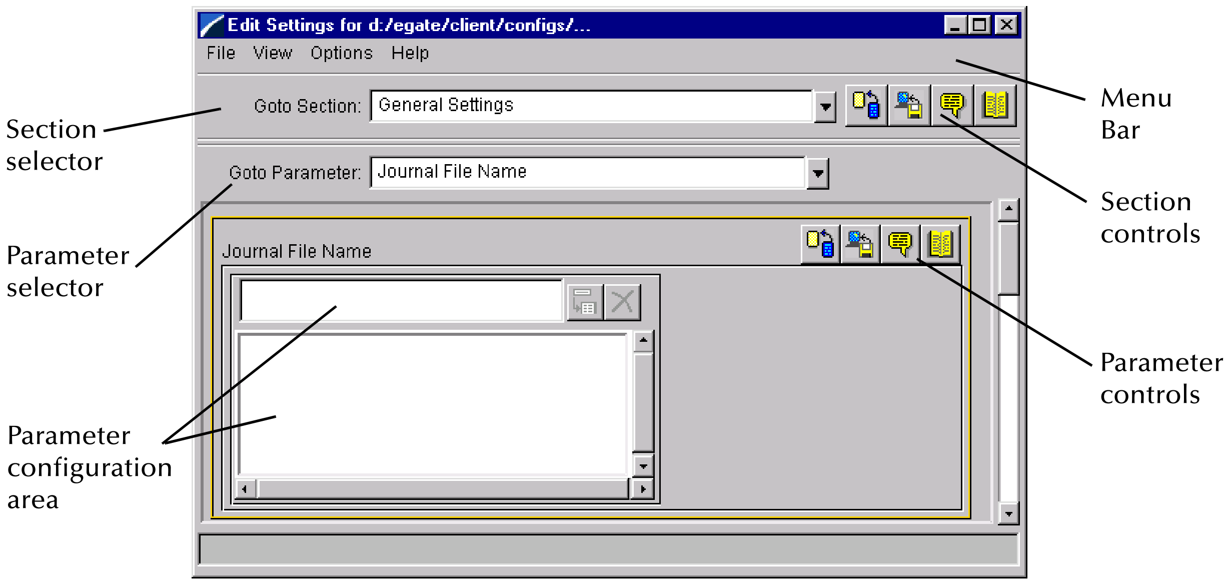
Configuration file:

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 now are in the e*Way Configuration Editor.

## Using the e*Way Editor

**Figure 28** 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.

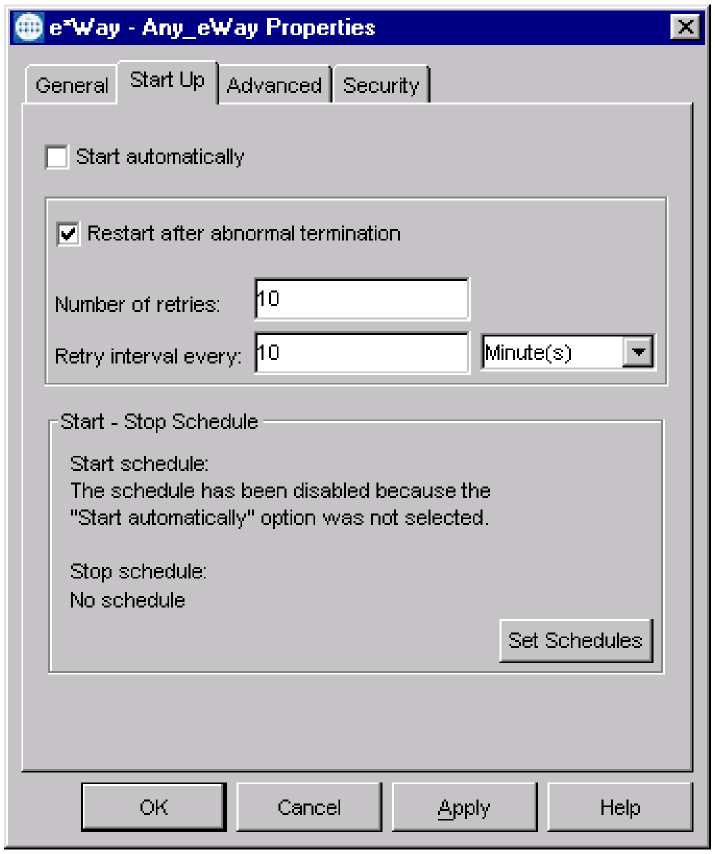
### 4.2.5 Setting Startup Options or Schedules

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

**Figure 29** e*Way Properties (Start-Up Tab)



☐ Start automatically

☑ Restart after abnormal termination

Number of retries: 10

Retry interval every: 10 Minute(s)

Start - Stop Schedule

Start schedule:
The schedule has been disabled because the "Start automatically" option was not selected.

Stop schedule:
No schedule

**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**.

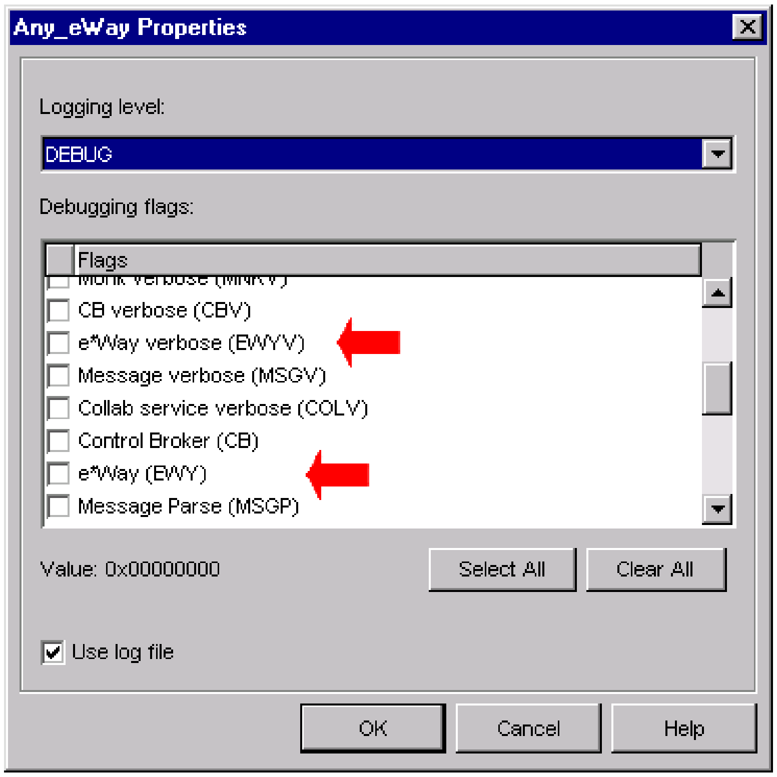
### 4.2.6 Activating or Modifying Logging Options

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**. The dialog window appears (see Figure 30).

**Figure 30** e*Way Properties (Advanced Tab - Log Option)



Any_eWay Properties

Logging level: DEBUG

Debugging flags:

Flags
- [ ] Monk verbose (MNKV)
- [ ] CB verbose (CBV)
- [ ] e*Way verbose (EWYV)
- [ ] Message verbose (MSGV)
- [ ] Collab service verbose (COLV)
- [ ] Control Broker (CB)
- [ ] e*Way (EWY)
- [ ] Message Parse (MSGP)

Value: 0x00000000

- [x] Use log file

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 routes it 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 (on Windows) or LD_LIBRARY_PATH (on UNIX) environmental variable includes a path to the PeopleSoft 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.

**Note:** *If you are using a non-English version of Siebel, you need to modify the Monk function* **siebel-eim-run-eim.monk**. *See* **Siebel IFB Functions** *on page 103.*

### 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.

***Note:*** *If you are running AIX 4.3.3, you should ensure that you have* **bos.rte.libpthreads** *version 4.3.3.51 or later. Earlier versions contain a recognized bug that can produce a memory leak. A patch is available from IBM.*

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.

Chapter 5

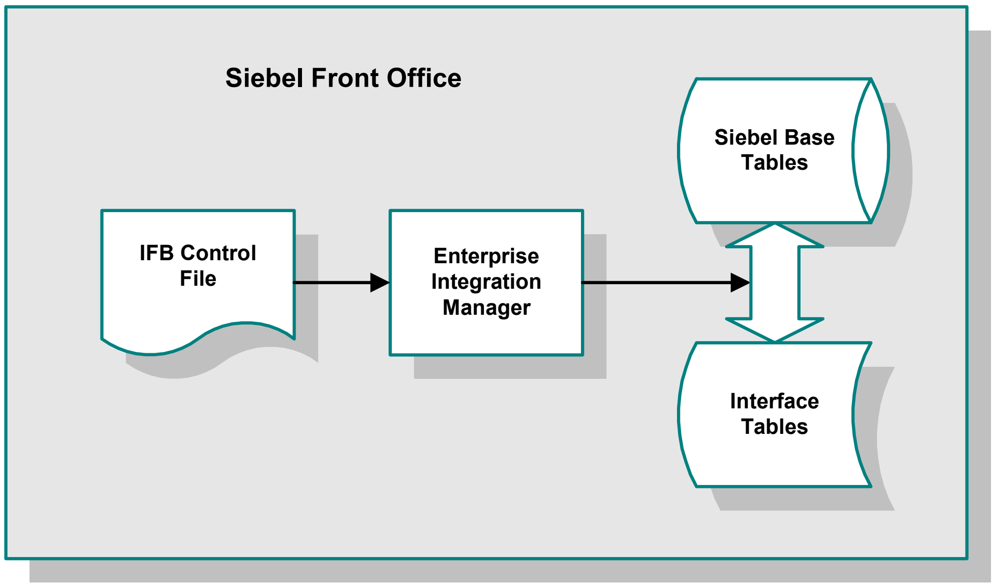
# Operational Overview

This chapter presents a brief overview of Siebel EIM, the Siebel-e*Way interface, and the architecture and basic internal processes of the Siebel EIM e*Way.

## 5.1 Siebel EIM

The Siebel Enterprise Integration Manager is the first step in exporting, and the last step in importing, data. The database consists of two main groups of tables, interface tables and base tables. The Siebel client application communicates directly with the highly normalized base tables. The interface tables are used as a staging area for importing, exporting, deleting and merging logical groups of data. Each interface table represents a subset of the data in a specific base table.

**Figure 31** Siebel Enterprise Integration Manager



When run, EIM coordinates the transfer of data between the base and interface tables. In addition, EIM creates and writes codes and row IDs to the base tables that directly correspond to the complex set of business rules used by the Siebel client application to display data. The EIM process must run on the server also running the Transaction Processor.

Typically, files to be imported have a predecessor-successor relationship, because Siebel requires files to be loaded in a certain manner. Thus any file depends on the completion of the preceding file. For example, if a project were receiving accounts, contacts and opportunities from an external source, they would receive either of the following:

- Three files from the external source
- One large file containing all of the pertinent information within one record

In scenario one, the three files would be Accounts, Contacts, and Opportunities, and Siebel requires the files be loaded in that order. Once EIM has completed interfacing Accounts, and the Accounts are loaded into the base tables, Contacts are loaded and an association is linked between the two entities. Opportunities follow the same logic.

In scenario two, the file is translated within e*Gate to appear as if three files were loaded into the correct interface tables. Both Scenario one and scenario two have the same predecessor-successor relationship that is inherent in Siebel. To load contacts successfully, the account that the contact belongs to must be loaded first.

See the *Siebel Administration Guide* for more details regarding Siebel EIM.

## 5.2 IFB Control File

A control file (***.ifb**) is used to determine what data types are loaded and how. The control file follows a certain format—it tells the interface how to log into the database and what process to run. It also lists the columns that the interface does *not* populate (in Siebel-inbound mode), thus preventing erroneous error messages. The IFB control file can be generated automatically by the e*Way or created manually, as configured by the user.

Following is an example of the control file format:

```
[Siebel Interface Manager]
        USER NAME = "SADMIN"
        PASSWORD = "SADMIN"
        TABLEOWNER = "SIEBEL"
        PROCESS = Import Both


 This group of processes provides samples for import data through
 all the interface tables, broken up into logical groups. Note
 that the order of import is often significant.

[Import Both]
TYPE = SHELL
INCLUDE = "Import Accounts"
INCLUDE = "Import Accounts wo SalesRep"


[Import Accounts]
        TYPE = IMPORT
        BATCH = 0111
        TABLE = S_ACCOUNT_IF
        ONLY BASE TABLES = S_ORG_EXT, S_ADDR_ORG, S_ORG_INDUST,
        S_ACCNT_POSTN
        INSERT ROWS = TRUE
        UPDATE ROWS = TRUE
        IGNORE BASE COLUMNS = S_ORG_EXT.PAYMENT_TERM_ID,
                         S_ORG_EXT.PAR_OU_ID,
                         S_ORG_EXT.PR_BL_ADDR_ID,
                         S_ORG_EXT.PR_SHIP_ADDR_ID

[Import Accounts wo SalesRep]
        TYPE = IMPORT
        BATCH = 0115
        TABLE = S_ACCOUNT_IF
        ONLY BASE TABLES = S_ORG_EXT, S_ADDR_ORG, S_ORG_INDUST
        INSERT ROWS = TRUE
        UPDATE ROWS = TRUE
        IGNORE BASE COLUMNS = S_ORG_EXT.PAYMENT_TERM_ID,
                         S_ORG_EXT.PAR_OU_ID,
                         S_ORG_EXT.PR_BL_ADDR_ID,
                         S_ORG_EXT.PR_SHIP_ADDR_ID
```

The IFB Control File is generated automatically by the **IFB Generation Function**, when the **Auto-Generate IFB File** parameter is selected. Otherwise, the **IFB Filename** indicated in the configuration file for the e*Way is used to process the data.

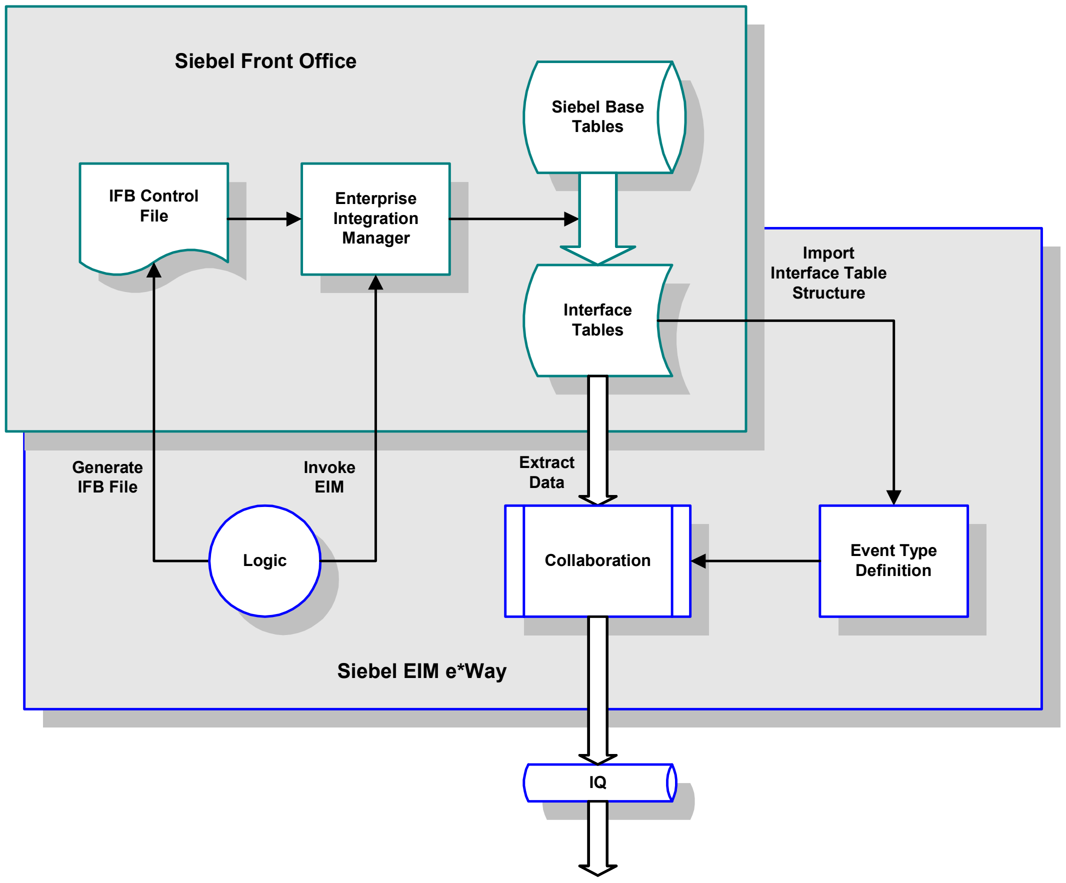
## 5.3 e*Way-Siebel Interaction

The Siebel-to-e*Gate operation is dealt with first, since it is less complex than the e*Gate-to-Siebel case.

### 5.3.1 Siebel to e*Gate

#### Process Flow

**Figure 32** Siebel-to-e*Gate Process Flow



1 Following a prescribed schedule, the Siebel EIM e*Way either generates or retrieves an IFB Control File, and sends it to Siebel.

2 The e*Way then invokes the Siebel EIM process.

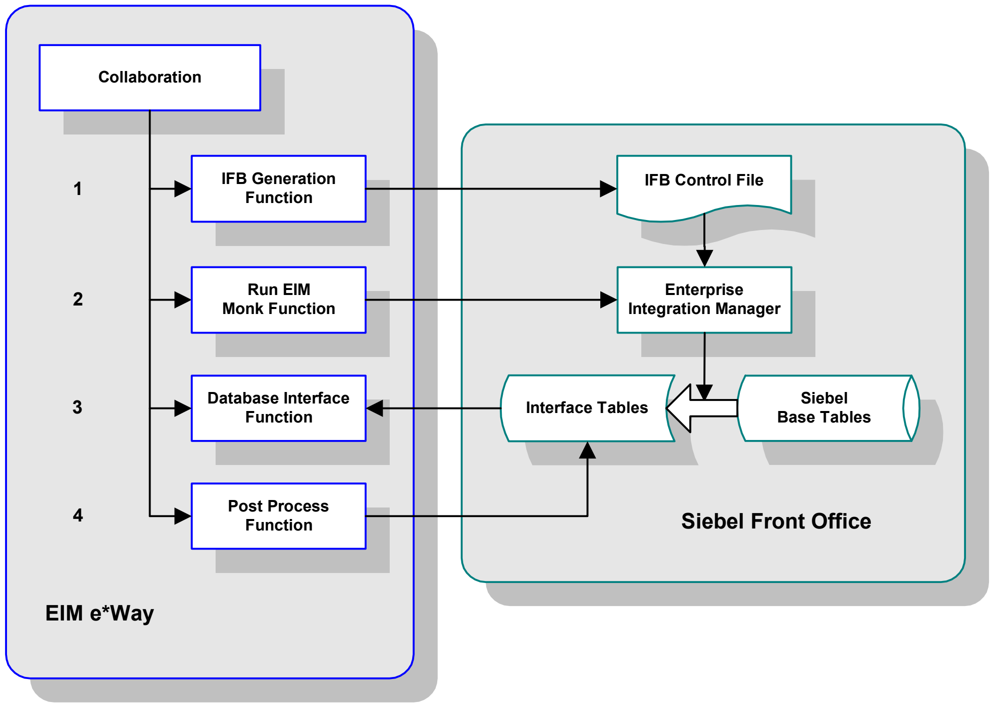
3 EIM then copies data from the Siebel Base Tables into the Interface Tables, as prescribed by the IFB Control File.

4 The Siebel EIM e*Way extracts the data from the interface tables and maps it into the correct Event Type Definition, following a pre-defined Collaboration.

5 The data is then passed to other e*Gate components for further processing (if required) and routing to the target application.

## Processing Logic

**Figure 33** Siebel-to-e*Gate Event Processing



### Creating the IFB Control File

If in the configuration file you have selected to **Auto-Generate IFB File**, then the IFB file is created based on the **IFB Generation Function**. Otherwise, the script uses the **IFB Filename** indicated in the configuration file for the e*Way to process the data. As part of generating the IFB file, the script determines if an export has been done previously (as stored in the **Last Time of Export File**. If no previous time is found, then the previous time of export is set to 01/01/1900.

### Transferring the Data using Siebel EIM

Following the IFB creation step, the script proceeds to invoke Siebel EIM, which replicates the Base Table data in the Interface Tables following the IFB Control File.

### Extracting the Data from the Interface Tables

The data is extracted from the Interface Tables using the **Database Interface Function**.

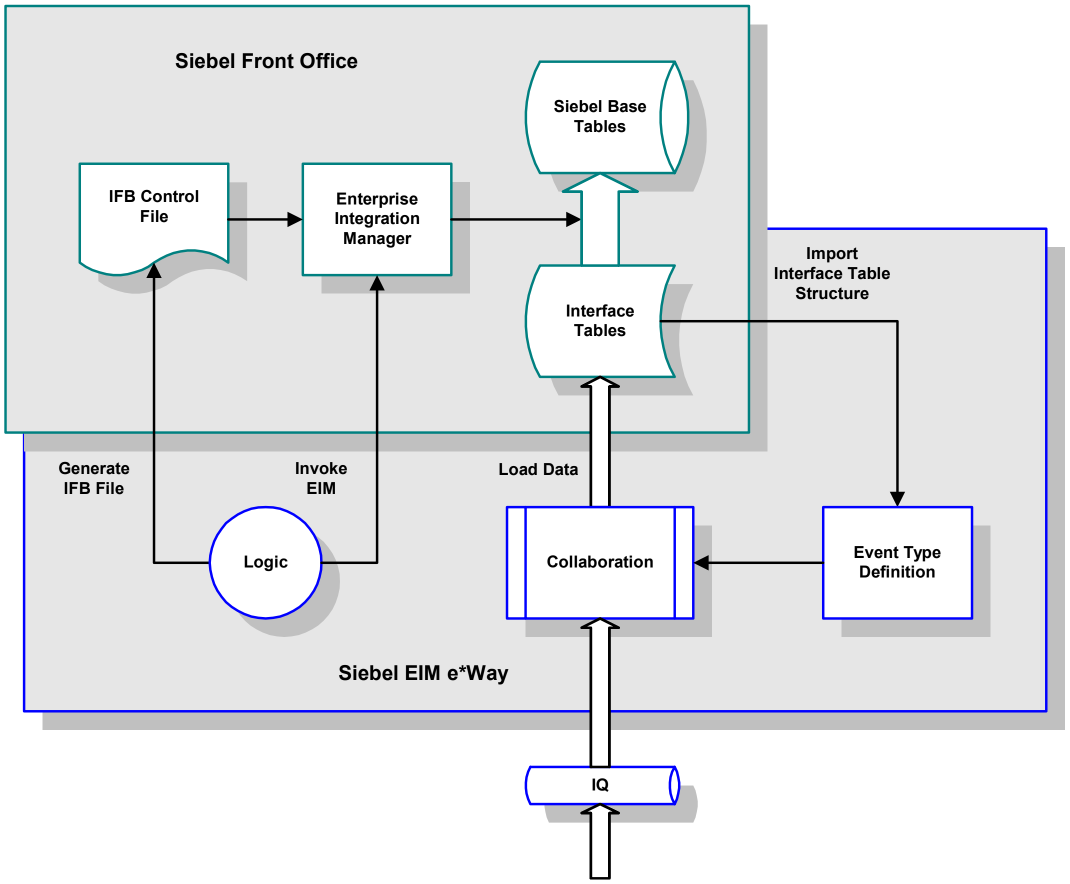
### Running the Post-Process Function

The script then invokes the **Post-Process Function**, which deletes all successfully-extracted records from the Interface Tables.

### 5.3.2 e*Gate to Siebel

#### Process Flow

**Figure 34** e*Gate-to-Siebel Process Flow



1 The Siebel EIM e*Way extracts data from an Intelligent Queue for processing.

2 The e*Way processes the information following a Collaboration previously created using structural information extracted from Siebel, and inserts validated rows into the Siebel Interface Tables.

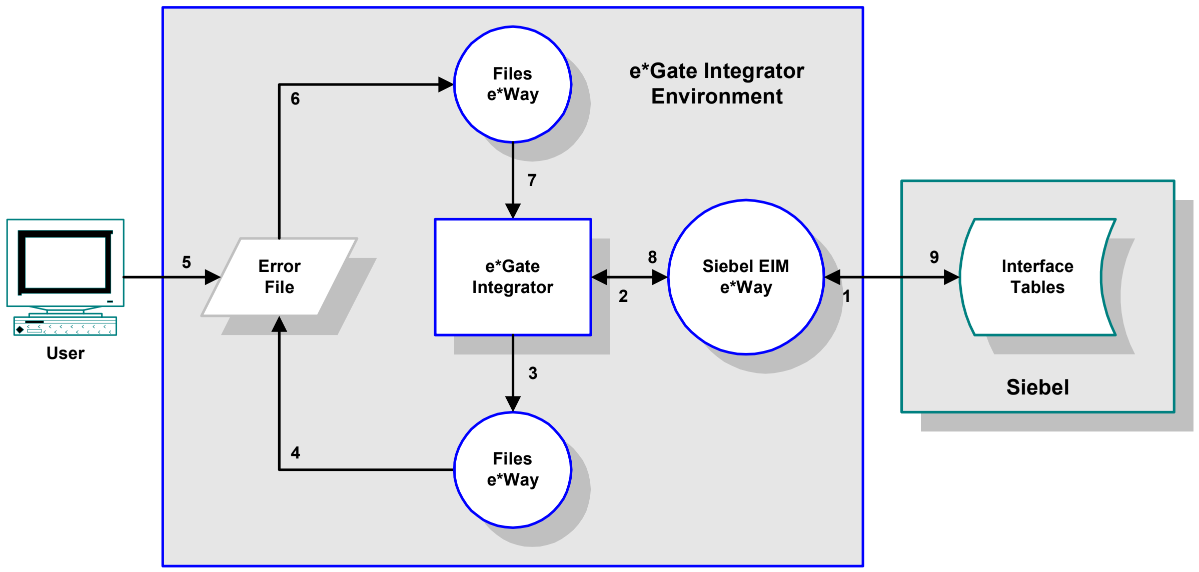
**Note:** *e*Gate allows the population of ROW_ID, IF_ROW_BATCH_NUM, and IF_ROW_STAT, according to the chosen naming standards during the insert statement.*

3 An IFB control file is built (or retrieved) and sent to the Siebel system for use by the EIM in populating the Base Tables with data from the Interface Tables.

4 The e*Way initiates the Siebel Enterprise Integration Manager (EIM) to load the data from the Interface Tables into the appropriate Siebel Base Tables.

5 Upon completion of the EIM process, e*Gate interrogates the **IF_ROW_STAT** column in the Siebel Interface Tables to verify the correct transfer of data to the Base Tables. All records transferred successfully are deleted from the Interface Tables.

**Figure 35** e*Gate-to-Siebel Error Processing

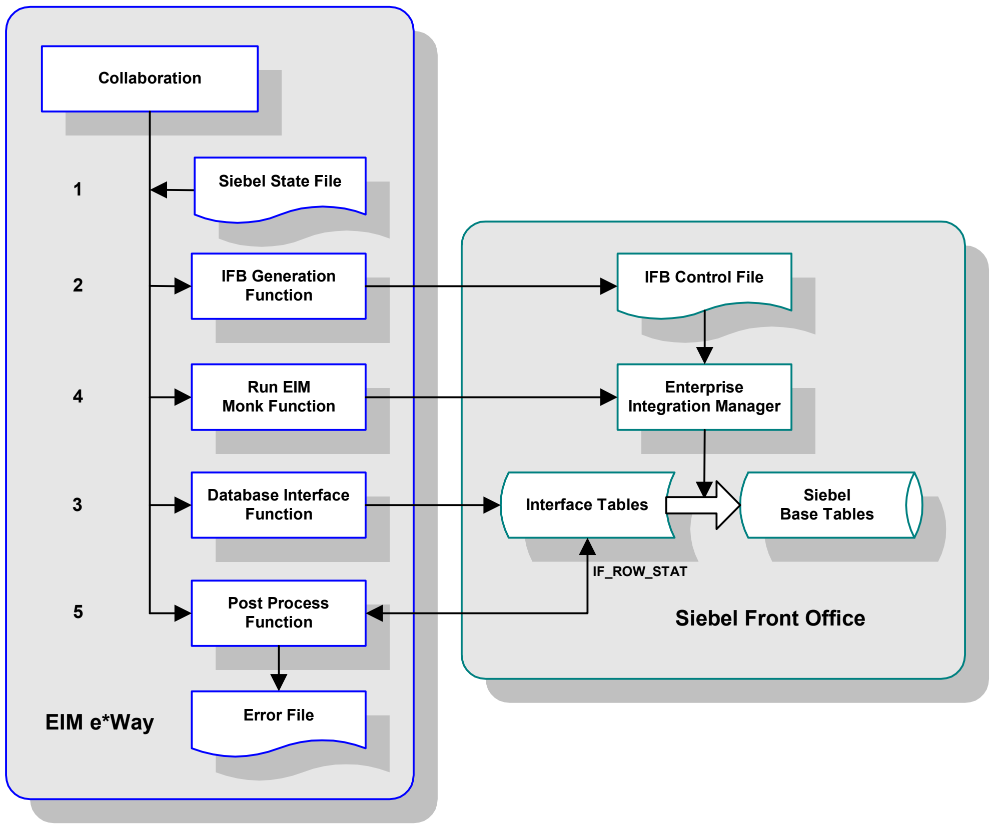


6 If any errors have occurred, the rejected records are pulled back into e*Gate by the EIM e*Way, placed in a error log file, and deleted from the Interface Tables.

7 User intervention is required to analyze and correct the data, and submit the records for reprocessing.

## Processing Logic

**Figure 36** e*Gate-to-Siebel Event Processing



### Checking the Event State

Upon start-up the Siebel e*Way checks the **Siebel State File** to determine the state of the inbound Event at the end of the last run. If no file is found the state is initialized to MSG_ACKED. There are three possible states:

- State 0 - MSG_ACKED - This is the initial state used for a new incoming Event. It also represents the final state in which the data has been inserted into the interface table, processed by the Enterprise Integration Manager, and an acknowledgment has been sent to e*Gate.
- State 1 - MSG_INSERTED - In this state the data has been inserted into the interface tables but no acknowledgment has been sent to e*Gate.
- State 2 - MSG_PROCESSED - In this state the data has been inserted into the interface table and processed by the Enterprise Interface Manager, but no acknowledgment has been sent to e*Gate.

### Creating the IFB Control File

If you have selected **Auto-Generate IFB File** in the configuration file, then the IFB file is created based on the **IFB Generation Function**. Otherwise, the script uses the **IFB Filename** indicated in the configuration file for the e*Way to process the data.

### Inserting the Data into the Interface Tables

If the state of the Event is **MSG_ACKED**, then the script proceeds to replicate the data in the Interface Tables using the **Database Interface Function**. If the state is other than **MSG_ACKED** it is set to **MONK_SKIP**, which instructs the script to skip this step. Once the records are successfully inserted into the Base Tables and committed, the state is changed to **MSG_INSERTED**.

### Transferring the Data using Siebel EIM

If the state of the Event is **MSG_INSERTED**, then the script proceeds to process the data using Siebel EIM. If the state is other than **MSG_ACKED** it is set to **MONK_SKIP**, which instructs the script to skip this step. Once the records are successfully transferred by EIM, the state is changed to **MSG_PROCESSED**.

### Running the Post-Process Function

The script then invokes the **Post-Process Function**, which extracts all records that were *not* successfully imported for this batch, and deletes *all* records from the table. When the Post-Process Function has completed successfully, the state is changed to **MSG_ACKED**.

### Error Correction

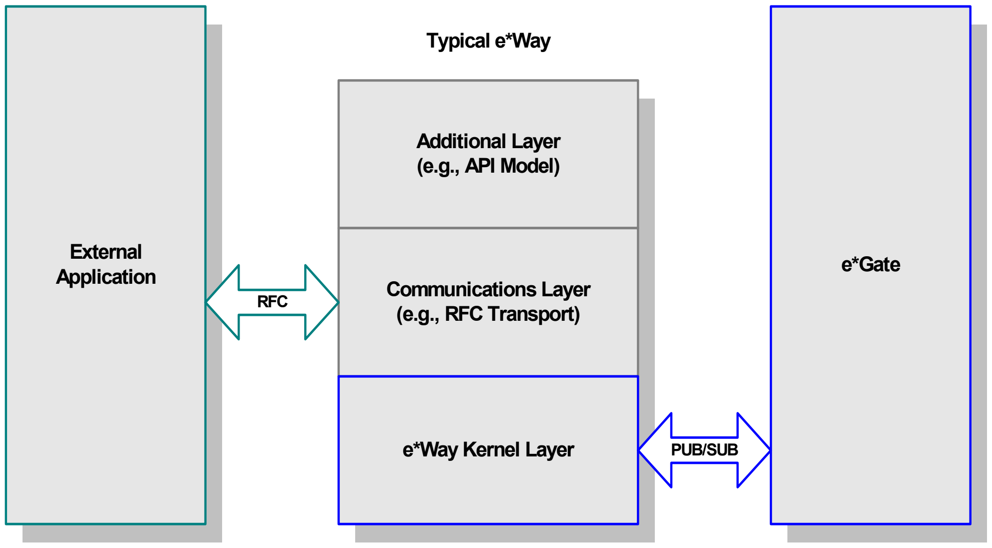
The script used to process the corrected error file works the same way as the one used originally, with the exception that it is expecting data in the Siebel IF Table format rather than the original input format. The difference is most apparent in the **Database Interface Function**.

***Note:*** *For descriptions of all Siebel functions, see* **Chapter 7**.

## 5.4 e*Way Architecture

Conceptually, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers (see Figure 37). 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 37** Typical e*Way Architecture

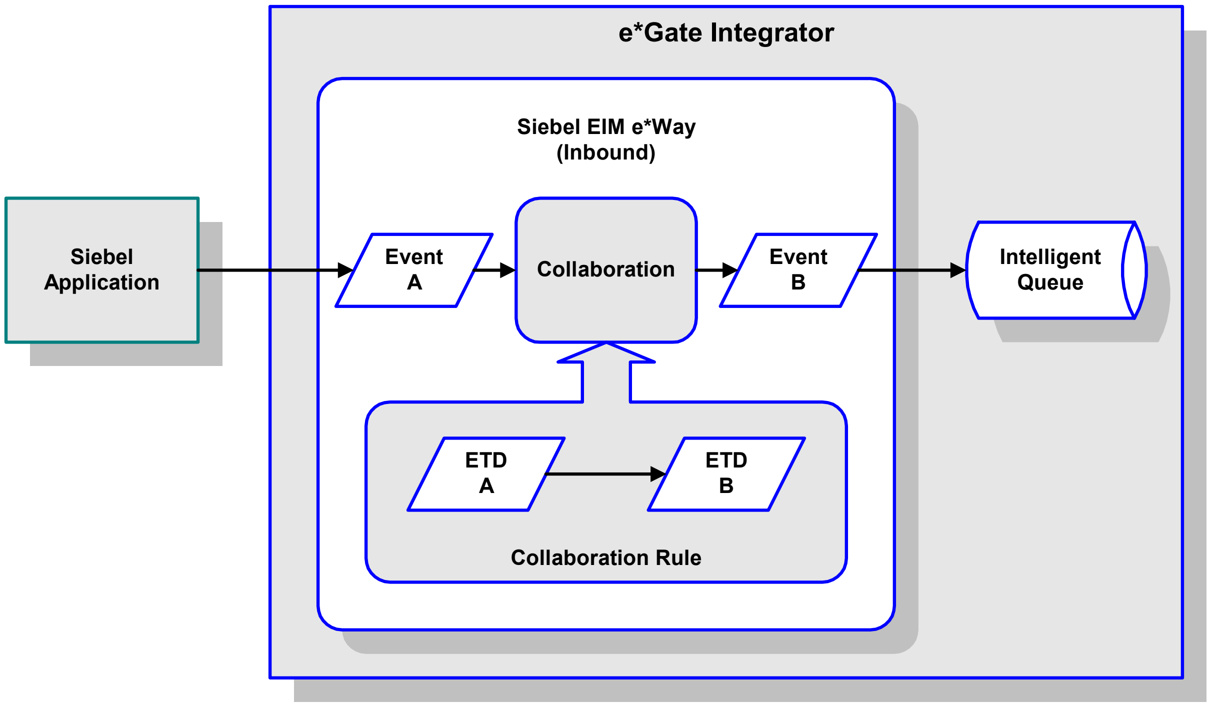

Typical e*Way
External Application
RFC
Additional Layer (e.g., API Model)
Communications Layer (e.g., RFC Transport)
e*Way Kernel Layer
PUB/SUB
e*Gate

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

**Figure 38** Collaborations and IQs



Configuration options that control the Monk environment and define the Monk functions used to perform various e*Way operations are discussed in **Chapter 6**. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft Word* or *Notepad*, or UNIX *vi*). The available set of e*Way API functions is described in **Chapter 7**. Generally, e*Way Kernel Monk functions should be called directly only when there is a specific need not addressed by higher-level Monk functions, and should be used only by experienced developers.

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

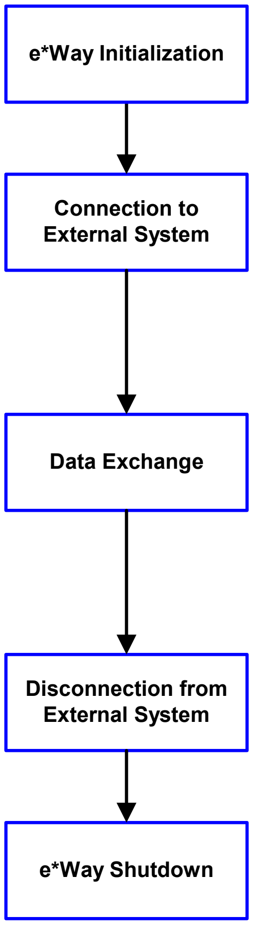
## 5.5 Basic e*Way Processes

**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 EIM 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
Connection to External System
Data Exchange
Disconnection from External System
e*Way Shutdown

Startup Function on page 88 (also see Monk Environment Initialization File on page 87)

External Connection Establishment Function on page 90
External Connection Verification Function on page 90

**Event-driven Data Exchange**
Process Outgoing Message Function on page 88

**Schedule-driven Data Exchange**
Exchange Data with External Function on page 89
Positive Acknowledgment Function on page 91
Negative Acknowledgment Function on page 92

External Connection Shutdown Function on page 91

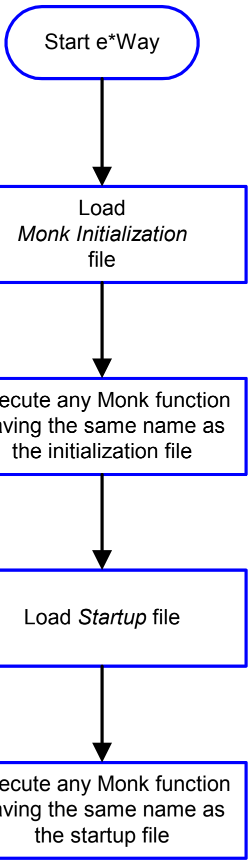
Shutdown Command Notification Function on page 93

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

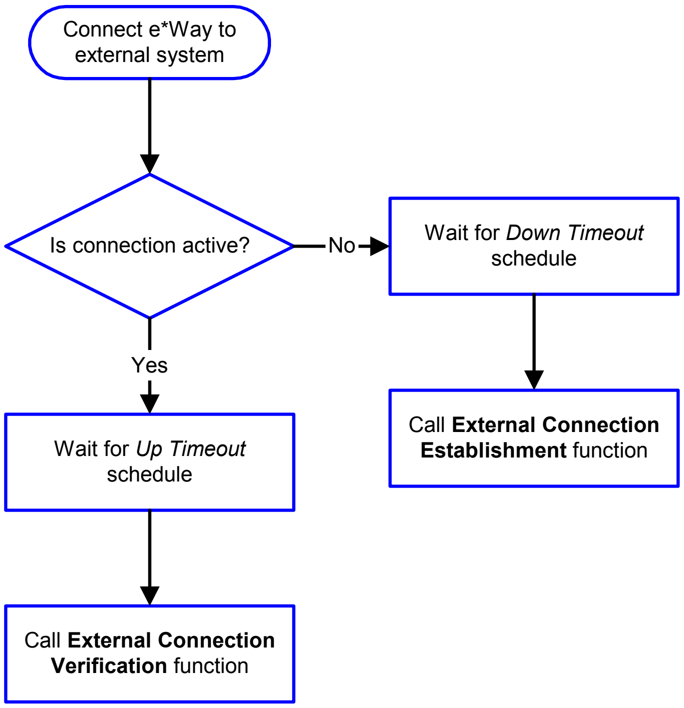
**Figure 39** Initialization Process


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

**Figure 40** Connection Process


Connect e*Way to external system
Is connection active?
No
Yes
Wait for *Down Timeout* schedule
Call **External Connection Establishment** function
Wait for *Up Timeout* schedule
Call **External Connection Verification** function

**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 42 on page 77** *and* **Figure 41 on page 76** *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 123 and* **send-external-down** *on page 123 for more information.*

## Data Exchange Process

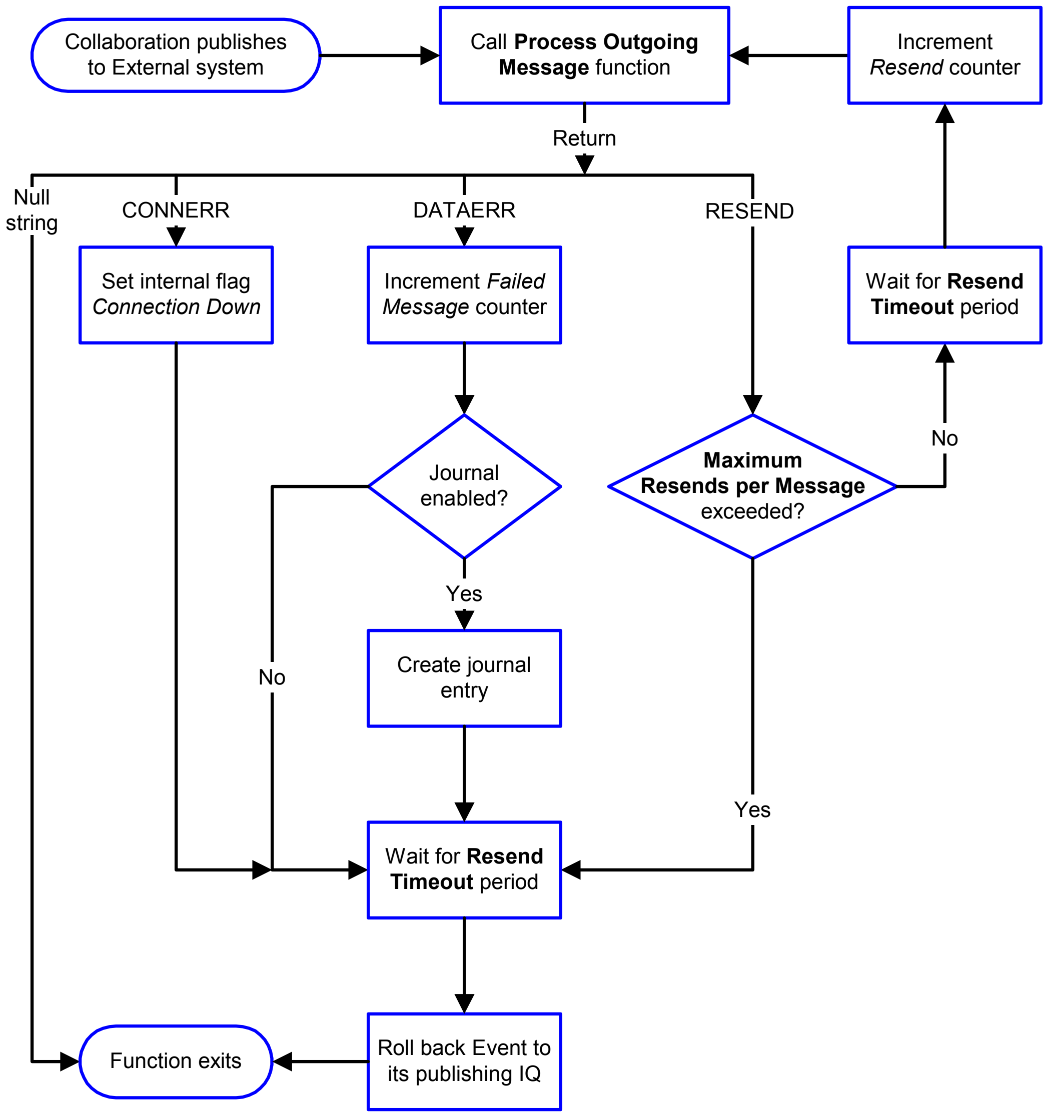
### Event-driven

Figure 41 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.

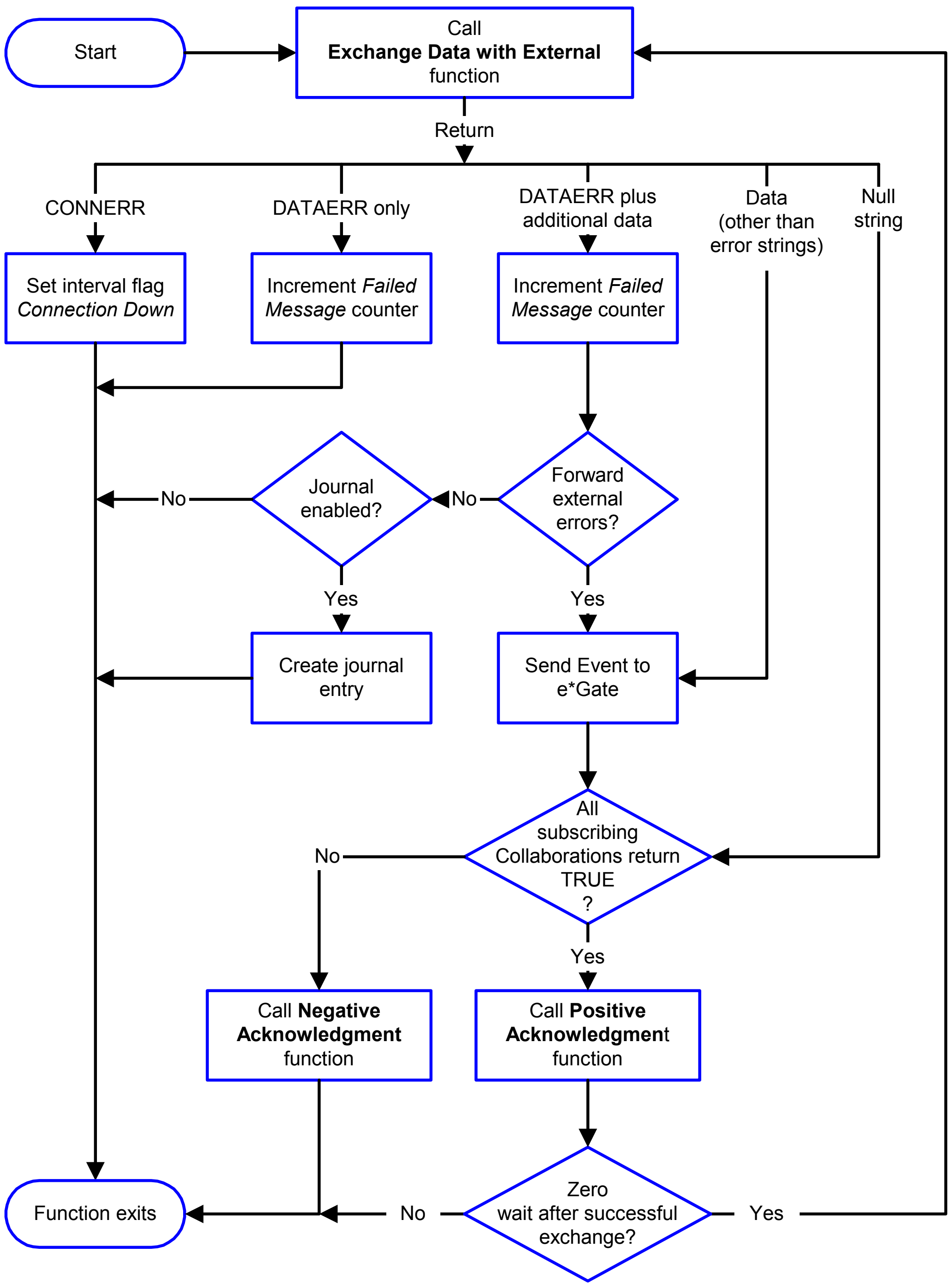
**Figure 41** Event-Driven Data Exchange Process


Collaboration publishes to External system
Call Process Outgoing Message function
Increment Resend counter
Return
Null string
CONNERR
DATAERR
RESEND
Set internal flag Connection Down
Increment Failed Message counter
Wait for Resend Timeout period
No
Journal enabled?
Maximum Resends per Message exceeded?
Yes
Create journal entry
No
Yes
Wait for Resend Timeout period
Function exits
Roll back Event to its publishing IQ

### Schedule-driven

Figure 42 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 42** Schedule-Driven Data Exchange Process


Start
Call Exchange Data with External function
Return
CONNERR
DATAERR only
DATAERR plus additional data
Data (other than error strings)
Null string
Set interval flag Connection Down
Increment Failed Message counter
Increment Failed Message counter
Journal enabled?
No
No
Forward external errors?
Yes
Yes
Create journal entry
Send Event to e*Gate
All subscribing Collaborations return TRUE ?
No
Yes
Call Negative Acknowledgment function
Call Positive Acknowledgment function
Function exits
No
Zero wait after successful exchange?
Yes

*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

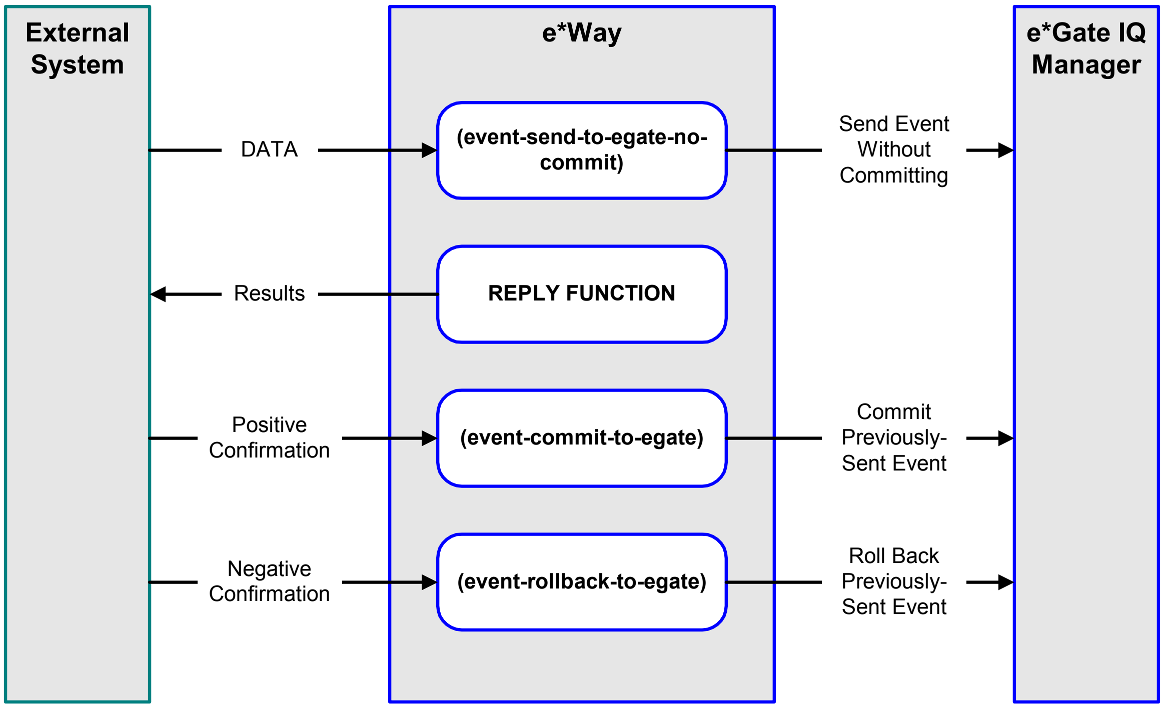
*Send Events to e*Gate* can be implemented using any of the following Monk functions:

- **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 43):

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

**Figure 43** Send Event to e*Gate with Confirmation

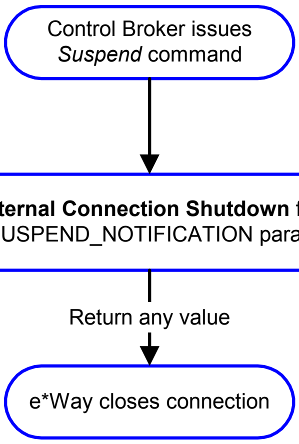

External System
e*Way
e*Gate IQ Manager
DATA
(event-send-to-egate-no-commit)
Send Event Without Committing
Results
REPLY FUNCTION
Positive Confirmation
(event-commit-to-egate)
Commit Previously-Sent Event
Negative Confirmation
(event-rollback-to-egate)
Roll Back Previously-Sent Event

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

## Disconnect from External Process

Figure 44 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

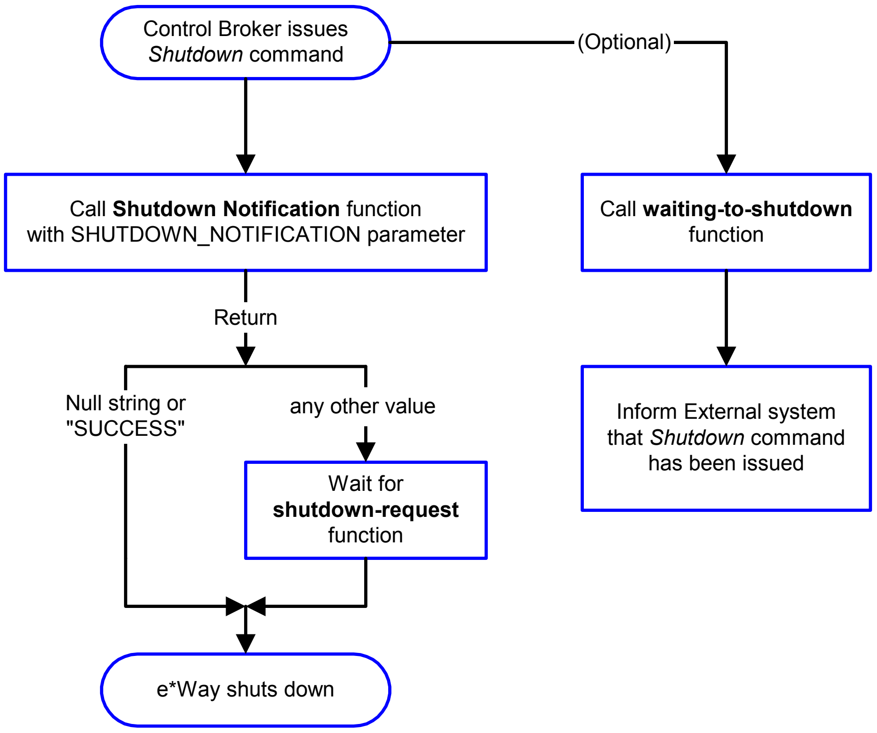
**Figure 44** Disconnect Process


Control Broker issues *Suspend* command
Call **External Connection Shutdown** function with SUSPEND_NOTIFICATION parameter
Return any value
e*Way closes connection

## Shutdown Process

Figure 45 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

**Figure 45** Shutdown Process


Control Broker issues *Shutdown* command
(Optional)
Call **Shutdown Notification** function with SHUTDOWN_NOTIFICATION parameter
Call **waiting-to-shutdown** function
Return
Null string or "SUCCESS"
any other value
Wait for **shutdown-request** function
Inform External system that *Shutdown* command has been issued
e*Way shuts down

Chapter 6

# Configuration Parameters

This chapter describes the configuration parameters for the Siebel EIM 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 52 for procedural information. The default configuration is provided in **SiebelEim.def**. The Siebel EIM e*Way's configuration parameters are organized into the following sections:

**General Settings** on page 81

**Communication Setup** on page 83

**Monk Configuration** on page 86

**Siebel Server** on page 94

**Siebel System Functions** on page 97

**Siebel System Import** on page 99

**Siebel System Export** on page 101

## 6.2 General Settings

The General Settings control top level operational parameters.

### Journal File Name

#### Description

Specifies the name of the journal file.

#### Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is stored in the e*Gate **SystemData** directory. See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

#### Additional Information

An Event is journaled for the following conditions:

- When the number of resends is exceeded (see **Max Resends Per Message**, below)
- When its receipt is due to an external error, but **Forward External Errors** is set to **No**

### 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. See **Exchange Data with External Function** on page 89 for more information.

### Required Values

**Yes** or **No**. The default value, **No**, specifies that error messages are not to be forwarded. See **Data Exchange Process** on page 76 for more information about how the e*Way uses this function.

## 6.3 Communication Setup

The Communication Setup parameters control the schedule by which the e*Way obtains data from the external system.

**Note:** *The schedule that you set using the e*Way's properties in the e*Gate Enterprise Manager controls when the e*Way executable runs. The schedule that you set within the parameters discussed in this section (using the e*Way Editor) determines when data are exchanged. Be sure that you set the "exchange data" schedule to fall within the "run the executable" schedule.*

### Start Exchange Data Schedule

#### Description

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

#### Required Values

One of the following:

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

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

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

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

#### Additional Information

When the schedule starts, the e*Way determines whether or not it is waiting to send an ACK or NAK to the external system (using the **Positive Acknowledgment Function** and **Negative Acknowledgment Function**) 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, this latter function is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

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).

See also **Zero Wait Between Successful Exchanges** on page 85 for more information.

## Stop Exchange Data Schedule

### Description

Establishes the schedule to stop data exchange.

### Required Values

One of the following:

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

  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 setting of this function is ignored and the e*Way invokes the **Exchange Data with External Function** immediately.

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

## 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 a message to the external system, after receiving an error message from the external system.

### Required Values

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

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 are 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/ewsiebeleim**.

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

## Monk Environment Initialization File

### Description

Specifies a file that contains environment initialization functions, which is loaded after the **Auxiliary Library Directories** are loaded. Any environment initialization functions called by this file accept no input, and must return a string.

### Required Values

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

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

### Returns

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

### Additional information

- Use this feature to initialize the e*Way's Monk environment (for example, to define Monk variables that are used by the e*Way's function scripts); it is good practice to initialize any global Monk variables that may be used by any other Monk Extension scripts
- The internal function that loads this file is called once when the e*Way first starts up
- The e*Way loads this file and try to invoke a function of the same base name as the file name

## Startup Function

### Description

Specifies a Monk function that the e*Way loads and invokes upon startup or whenever the e*Way's configuration is reloaded. It is called after the e*Way loads the specified **Monk Environment Initialization File** and any files within the specified **Auxiliary Library Directories**. This function accepts no input, and must return a string.

This function should be used to initialize the external system before data exchange starts.

### Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **siebel-eim-startup**.

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

### Returns

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

## Process Outgoing Message Function

### Description

Specifies the Monk function responsible for sending outgoing messages (Events) from the e*Way to the external system. This function is event-driven, rather than schedule-driven). The function requires a non-null string as input (i.e., the outgoing Event to be sent), and must return a string.

### Required Values

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

### Required Values

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

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

### Returns

- A *null string* (“”) indicates that the data exchange was completed successfully, but with no resultant data sent back to the e 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 (repeatedly) when it has determined that the connection to the external system is down. The function accepts no input and must return a string.

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

### Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **siebel-eim-external-connection-establishment**.

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

### Returns

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

## External Connection Verification Function

### Description

Specifies a Monk function that the e*Way calls when its internal variables show that the connection to the external system is up. It is executed according to the interval specified within the **Up Timeout** parameter, and is called *only* according to this schedule. The function accepts no input and must return a string.

### Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **siebel-eim-verify-db-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

If this function is not specified, the e*Way executes the **External Connection Establishment Function** in its place. This latter function also is called when the e*Way has determined that its connection to the external system is down.

## External Connection Shutdown Function

### Description

Specifies a Monk function that the e*Way calls to shut down the connection to the external system. This function is invoked only when the e*Way receives a *suspend* command from a Control Broker.

### Required Values

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

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

### Input

A string indicating the purpose for shutting down the connection.

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

### Returns

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

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

## Positive Acknowledgment Function

### Description

This function is loaded during the initialization process and is called when all data received from the external system has been processed and enqueued successfully.

### Required Values

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

### Input

A string, the inbound Event to e*Gate.

### Returns

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

### Additional Information

- After the **Exchange Data with External Function** returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. The e*Way executes this function only if the Event's processing is completed successfully by *all* the Collaborations to which it was sent; otherwise, the e*Way executes the **Negative Acknowledgment Function**.
- This function can return data to be queued, but the e*Way will *not* acknowledge the data with an **ACK** or **NAK**.

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

## Negative Acknowledgment Function

### Description

This function is loaded during the initialization process and is called when the e*Way fails to process or enqueue data received from the external system successfully.

### Required Values

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

### Input

A string, the inbound Event to e*Gate.

### Returns

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

### Additional Information

- This function is called only during the processing of inbound Events. After the **Exchange Data with External Function** returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. The e*Way executes this function if the Event's processing is not completed successfully by *all* the Collaborations to which it was sent; otherwise, the e*Way executes the **Positive Acknowledgment Function**.
- This function can return data to be queued, but the e*Way will *not* acknowledge the data with an **ACK** or **NAK**.

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

## Shutdown Command Notification Function

### Description

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

### Required Values

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

The parameters in this section initialize the Siebel Server variables.

### Database Client Connectivity

#### Description

Specifies the connection type version used to connect to the database.

#### Required Values

One of the following options:

- ODBC
- ORACLE7
- ORACLE8
- ORACLE8i
- SYBASE

The default is **ODBC**.

### Database Host Name

#### Description

Specifies the host machine where the Siebel database resides. If using ODBC, this is the data source name corresponding to the Siebel database.

#### Required Values

A string.

### Database Table Owner

#### Description

Specifies the Siebel database logon name that owns the tables being operated on.

#### Required Values

A string.

### Database User Name

#### Description

Specifies the Siebel database logon ID.

#### Required Values

A string.

---

### Database Password

#### Description

Specifies the Siebel database password.

#### Required Values

An encrypted string.

---

### Gateway Server

#### Description

Specifies the address or name of the Gateway Server machine.

#### Required Values

A string.

---

### Enterprise Server Name

#### Description

Specifies the Enterprise Server name.

#### Required Values

A string.

---

### Administrator User Name

#### Description

Specifies the Administrator user name.

#### Required Values

A string.

---

### Administrator Password

#### Description

Specifies the Administrator password.

#### Required Values

An encrypted string.

### Name

#### Description

Specifies the name of the Siebel Server you are using.

#### Required Values

A string.

### Path to Siebel Server

#### Description

Specifies the fully qualified path to the home of the Siebel Server.

#### Required Values

A pathname, for example: \\**SiebelHost****Siebel****...**

## 6.6 Siebel System Functions

The functions and parameters in this section are used in processing Siebel data.

### Batch Number

#### Description

Specifies the batch number to be used in the IFB file and with processing the data.

#### Required Values

A number.

### IFB Filename

#### Description

Specifies the name of the IFB file.

#### Required Values

A filename.

#### Additional Information

If your IFB file resides on the same host as the Siebel server, you can reference the filename either relative to the e*Gate path on the Siebel server host, or use an absolute path with the drive letter.

If your IFB file does not reside on the same host as the Siebel server, you must give the Siebel server the fully-qualified path name of the directory and also share the directory so that the Siebel server host has access to it.

#### Example

If the Siebel server resides on a computer called **Sieb_Box** and you want to store the IFB file (**s_account_if.ifb**) on another computer, **IFB_Host**, in the directory path **C:\egate\client\ifb**, you must share this directory by giving read (or higher) permission to the Siebel server user.

For this configuration the IFB filename would be \\**IFB_Host\ifb\s_account_if.ifb**.

### Auto-Generate IFB File

#### Description

Specifies whether or not the e*Way generates the IFB file based on the **IFB Generation Function**.

#### Required Values

**Yes** or **No**.

If **Yes** is selected, this e*Way generates the IFB file based on the IFB Generation Function.

If **No** is selected, this e*Way obtains the IFB file from the name given for the IFB Filename parameter.

## IFB Generation Function

### Description

Generates the IFB file. See **IFB Control File** on page 64 for more information.

### Required Values

A string.

## Database Interface Function

### Description

Used to interface with the database.

### Required Values

A string.

### Additional Information

A function to import data would involve inserting the data into the interface table, committing the data to the database, and then processing the data. A function to export data would involve fetching the data and sending the data to e*Gate.

## Post-Process Function

### Description

Use of this function pertains to any cleanup work.

### Required Values

A string.

### Additional Information

For example, upon importing data the post-process function might delete all successfully imported records and retrieve all records not successfully imported.

## 6.7 Siebel System Import

The parameters in this section initialize the Siebel Server variables for Import only.

### Continue on Insert Error

#### Description

Specifies whether or not the e*Way continues to insert the remaining records in the batch into the interface table upon a **db-struct-insert** error.

#### Required Values

**Yes** or **No**.

If **Yes** is selected, the e*Way continues to insert the remaining records in the batch into the interface table upon a **db-struct-insert** error.

If **No** is selected, the e*Way rolls back records already inserted into the interface table and return "**FAILURE**" to e*Gate.

### Siebel State File

#### Description

Determines the state of the inbound message at the end of the last run.

#### Required Values

None.

#### Additional Information

Upon startup the Siebel e*Way checks this file. If no file is found, the state is initialized to **MSG_ACKED**. The Siebel State File accepts a relative path (e.g., **data\siebel.doc**) relative to **eGate\client**.

There are three possible states:

**State 0 - MSG_ACKED**

This is the initial state used for new incoming messages. It also represents the final state in which the data has been inserted into the interface table, processed by the Enterprise Interface Manager and the next message is processed into the queue.

**State 1 - MSG_INSERTED**

In this state the message has been inserted into the interface table but no acknowledgment has been sent to e*Gate

**State 2 - MSG_PROCESSED**

In this state the message has been inserted into the interface table and processed by the Enterprise Interface Manager, but no acknowledgment has been sent to e*Gate.

**Note:** *The integrity of this file is crucial to maintaining the correct state of the import; therefore, editing of this file is not recommended.*

## 6.8 Siebel System Export

The parameters in this section initialize the Siebel Server parameters for Export only.

### Last Time of Export File

#### Description

Specifies the file that holds the time of when the last export was done.

#### Required Values

None.

#### Additional Information

The integrity of this file is crucial to maintaining the correct time of when the last export was done in the Siebel e*Way; therefore, editing of this file is not recommended.

### Siebel State File

#### Description

Determines the state of the outbound Event at the end of the last run.

#### Required Values

None.

#### Additional Information

The Siebel State File accepts a relative path (e.g., **data\siebel.doc**) relative to **eGate\client**.

***Note:*** *The integrity of this file is crucial to maintaining the correct state of the import; therefore, editing of this file is not recommended.*

Chapter 7

# API Functions

This chapter describes the various Monk functions used by the SeeBeyond e*Way Intelligent Adapter for Siebel EIM.

## 7.1 Overview

The Siebel EIM e*Way's functions are categorized as follows:

## 7.2 Siebel IFB Functions

The following functions are used to create the IFB file for use by Siebel EIM, and to run EIM using the IFB file.

### siebel-eim-create-ifb

#### Description

Takes the IFB event-structure and creates the IFB file used by Siebel EIM.

#### Signature

```
(siebel-eim-create-ifb <event-structure>)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| event-structure | string | Event-structure to be used to create the IFB file |

#### Returns

Upon completion, the string “**done\n**”.

#### Throws

None.

#### Location

**siebel-eim-create-ifb.monk**

### siebel-eim-create-ifb_entry

#### Description

Called from within **siebel-eim-create-ifb** to format the file being created.

#### Signature

```
(siebel-eim-create-ifb_entry <event-structure>)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| event-structure | string | Event-structure to be formatted to create the IFB file |

#### Returns

None.

#### Throws

None.

### Location

**siebel-eim-create-ifb.monk**

## siebel-eim-run-eim

### Description

Runs the Siebel Enterprise Integration Manager using the appropriate configuration (IFB) file.

### Signature

```
(siebel-eim-run-eim)
```

### Parameters

None.

### Returns

Depends on status returned by EIM. The following string values are possibilities:

- "MONK_FAILURE"
- "MONK_SUCCESS"
- "MONK_SHUTDOWN"

### Throws

None.

### Additional Information

If you are using a non-English version of Siebel, you need to modify this function. The following statements should be changed to reflect the language-specific task states for your system:

```
(string=? "Error" task-state)
(string=? "Completed" task-state)
(string=? "Exited with error" task-state)
```

If the current task state is not equal to any of these strings then the function assumes the task is still running.

### Location

**siebel-eim-run-eim.monk**

## 7.3 Siebel Server Connection Functions

The following functions are used to connect with the Siebel server.

### siebel-eim-server-capture

#### Description

Runs **srvrcmd** on the Siebel EIM using **srvrmgr.exe**. The output generated by **srvrmgr.exe** is captured and stored in the **srvrOutput** Event Type Definition.

#### Signature

```
(siebel-eim-server-capture srvcmd srvrOutput)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| srvcmd | string | A command used to invoke EIM. |
| srvrOutput | string | A string storing the information captured by srvrgr.exe |

#### Returns

A **string** containing the information returned.

If there is a failure on the part of Siebel EIM, then a **-1** is stored in:

```
~srvrOutput%Siebel_EIM_server_tasks.ErrorReturn.value
```

and an error string is stored in:

```
~srvrOutput%Siebel_EIM_server_tasks.ErrorString.value.
```

#### Throws

None.

#### Location

**siebel-eim-server-capture.monk**

### siebel-eim-server-start-task

#### Description

Starts an EIM task using **ifbfile**.

#### Signature

```
(siebel-eim-server-start-task <ifbfile>)
```

### Parameter

| Name | Type | Description |
|---|---|---|
| ifbfile | string | The IFB file used to start an EIM task. |

### Returns

If successful, an integer representing the task number. If unsuccessful, the integer **0**.

### Throws

None.

### Location

**siebel-eim-server-capture.monk**

## siebel-eim-server-task-state

### Description

Queries Siebel EIM and returns the task state for the specified **task-num**.

### Signature

```
(siebel-eim-server-task-state <task-num>)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| task-num | integer | Specifies the task number in question. |

### Returns

If successful, a **string** representing the task state for the specified **task-num**. Upon failure, the string **"Error:"** followed by the reason.

### Throws

None.

### Location

**siebel-eim-server-capture.monk**

## siebel-eim-server-task-status

### Description

Queries Siebel EIM and returns the task status for the specified **task-num**.

### Signature

```
(siebel-eim-server-task-status <task-num>)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| task-num | integer | Specifies the task number in question. |

### Returns

If successful, a **string** representing the task state for the specified **task-num**. Upon failure, the string **"Error:"** followed by the reason.

### Throws

None.

### Location

**siebel-eim-server-capture.monk**

## 7.4 Siebel Utility Functions

The following functions are used as utilities by the Siebel EIM e*Way.

### stripout-string

#### Description

Returns the **string** with the **pattern** removed.

#### Signature

```
(stripout-string <pattern> <string>)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| pattern | string | The pattern to be removed. |
| string | string | The string in which the pattern is to be removed. |

#### Returns

Returns a **string** with the requested pattern removed.

#### Throws

None.

#### Location

**siebel-eim-utils.monk**

### db-systime

#### Description

Queries the database for the current timestamp, for the given **connection-handle**.

#### Signature

```
(db-systime <connection-handle>)
```

#### Parameters

| Name | Type | Description |
|---|---|---|
| connection-handle | string | Specifies the connection-handle for which the timestamp from the database is requested |

#### Returns

Returns the **timestamp** requested from the database.

### Throws

None.

### Location

**siebel-eim-utils.monk**

## siebel-eim-get-last-exported-time

### Description

Returns the time found in the **siebel-eim-last-exported-time-file**.

### Signature

```
(siebel-eim-get-last-exported-time)
```

### Parameters

None.

### Returns

A **string** indicating the time (the filename is determined from the configuration file).

Upon error, a Boolean false (**#f**).

### Throws

None.

### Location

**siebel-eim-utils.monk**

## siebel-eim-set-last-exported-time

### Description

Updates both the **siebel-eim-import-state-file** and **siebel-eim-import-state** with the time of the last export of data.

### Signature

```
(siebel-eim-set-last-exported-time <timestring>)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| timestring | string | A string containing the time the last export was performed. |

### Returns

None.

### Throws

None.

### Location

**siebel-eim-utils.monk**

## siebel-eim-get-import-state

### Description

Returns the state found in the **siebel-eim-import-state-file**. The filename is determined from the configuration file.

### Signature

```
(siebel-eim-get-import-state)
```

### Parameters

None.

### Returns

A **string** indicating the state; for example, **"Couldn't locate"** or **"OK, state will be set to 0 - MSG_ACKED"**.

### Throws

None.

### Location

**siebel-eim-utils.monk**

## siebel-eim-set-import-state

### Description

Updates both **siebel-eim-import-state-file** and **siebel-eim-import-state** with **statestring.**

### Signature

```
(siebel-eim-set-import-state <statestring>)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| statestring | string | A numeric value indicating the current state. |

### Returns

None.

### Throws

None.

### Location

**siebel-eim-utils.monk**

## return-path

### Description

Returns the path name originating at a defined point.

### Signature

```
(return-path <string>)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| string | string | A string containing information allowing for retrieval of the table path. |

### Returns

Upon success, a **string** representing everything following the backslash (\), in order to obtain the table path; upon failure, an **empty string**.

### Throws

None.

### Location

**siebel-eim-utils.monk**

## 7.5 e*Way Initialization Functions

The following functions are used by the Siebel EIM e*Way to perform the initialization process.

### siebel-eim

#### Description

Prepares the initialization process for the e*Way by loading the **stc_monkutils.dll** file.

#### Signature

```
(siebel-eim)
```

#### Parameters

None.

#### Returns

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

#### Throws

None.

#### Location

**siebel-eim.monk**

### siebel-eim-init

#### Description

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

#### Signature

```
(siebel-eim-init)
```

#### Parameters

None.

#### Returns

If successful, the string "**SUCCESS**"; otherwise, "**FAILURE**" and the e*Way shuts down.

#### Throws

None.

#### Location

**siebel-eim-init.monk**

## siebel-eim-startup

### Description

A sample Monk function for e*Way startup.

### Signature

```
(siebel-eim-startup)
```

### Parameters

None.

### Return Values

The string "SUCCESS".

### Throws

None.

### Location

## siebel-eim-shutdown

### Description

Called by the system to request that the external Siebel database connection shutdown. A return value of "SUCCESS" indicates that the shutdown can occur immediately, any other return value indicates that the shutdown event must be delayed. The user is then required to execute a **shutdown-request** call from within a monk function to allow the requested shutdown process to continue.

### Signature

```
(siebel-eim-shutdown <string>)
```

### Parameters

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

### Returns

If successful, the string "SUCCESS", which allows an immediate shutdown to occur. Anything else delays shutdown until the **shutdown-request** is executed successfully.

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-external-db-connection

### Description

Logs into the external connection

### Signature

```
(siebel-eim-verify-db-connect)
```

### Parameters

None.

### Returns

If connection to the database is successful, the string "**SUCCESS**"; otherwise, "**FAILURE**".

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-verify-db-connect

### Description

Obtains the status of the external system connection.

### Signature

```
(siebel-eim-verify-db-connect)
```

### Parameters

None.

### Returns

If successful, the string "**SUCCESS**"; otherwise, "**FAILURE**".

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-check-connect

### Description

Determines whether the connection to the external database is **UP** or **DOWN**.

### Signature

```
(siebel-eim-check-connect)
```

### Parameters

None.

### Returns

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

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-external-connection-establishment

### Description

A sample Monk function for establishing a connection with the external system.

### Signature

```
(siebel-eim-external-connection-establishment)
```

### Parameters

None.

### Returns

Upon success, the string “UP”; upon failure, the string “DOWN”.

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-ack

### Description

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

### Signature

```
(siebel-eim-ack <message-string>)
```

### Parameters

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

### Returns

**Success:**

An empty string indicates a successful operation.

**Failure:**

The string **“CONNERR”** indicates a loss of connection with the external, and client moves to a down state and attempts to connect. On reconnect, the **pos-ack** function is re-executed.

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-nack

### Description

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

### Signature

```
(siebel-eim-nack <message-string>)
```

### Parameters

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

### Returns

**Success:**

An empty string indicates a successful operation.

**Failure:**

The string **“CONNERR”** indicates a loss of connection with the external, and client moves to a down state and attempts to connect. On reconnect, the **pos-ack** function is re-executed.

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-process-outgoing-message

### Description

A sample Monk function for event-driven data exchange.

### Signature

```
(siebel-eim-process-outgoing-message <message-string>)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| message-string | string | The message string from the IQ. |

### Returns

An empty string (“”).

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-exchange-data

### Description

A sample Monk function for schedule-driven data exchange.

### Signature

```
(siebel-eim-exchange-data)
```

### Parameters

None.

### Returns

An empty string (“”).

### Throws

None.

### Location

**siebel-eim-init.monk**

## siebel-eim-return-empty-string

### Description

A sample Monk function for event-driven data exchange (obsolete).

### Signature

```
(siebel-eim-return-empty-string <message-string)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| message-string | string | The message string from the IQ. |

### Return Values

An empty string (“”).

### Throws

None.

### Location

**siebel-eim-init.monk**

***Note:*** *This function is obsolete—use* **siebel-eim-process-outgoing-message**.

## 7.6 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

#### 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 121

**event-send-to-egate-no-commit** on page 121

## 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 120

**event-send-to-egate-no-commit** on page 121

## 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 119

**event-rollback-to-egate** on page 120

**event-send-to-egate** on page 120

**event-send-to-egate-ignore-shutdown** on page 121

## 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 84

**Zero Wait Between Successful Exchanges** on page 85

## 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