SeeBeyond™ eBusiness Integration Suite

e*Way Intelligent Adapter for SWIFT ADK User's Guide

Release 4.5.3

SEEBEYOND™

Version 20030131054319.

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 the appropriate operating systems (see **Availability** on page 13)
- Windows-style GUI operations
- SWIFTAlliance and SWIFT ADK concepts and operations

P.2 Organization

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

The second part, consisting of Chapters 5-7, describes the architecture and internal functionality of the e*Way. This part should be of particular interest to a Developer involved in customizing the e*Way for a specific purpose. Information contained in this part that is necessary for the initial setup of the e*Way is cross-referenced in the first part of the guide, at the appropriate points in the procedures.

P.3 Nomenclature

Note that for purposes of brevity, the e*Way Intelligent Adapter for SWIFT is frequently referred to as the SWIFT ADK 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 in italics 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.

P.6 Additional Documentation

- Many of the procedures included in this User's Guide are described in greater detail in the *e*Gate Integrator User's Guide*.

Chapter 1

Introduction

This chapter provides a brief introduction to SWIFT and the e*Way Intelligent Adapter for SWIFT ADK.

1.1 SWIFT Overview

The Society for World-wide Interbank Financial Telecommunication (SWIFT) is a bank-owned cooperative which supplies secure payment event transfer, matching, and other services to owner/member banks and other financial organizations (including brokers, securities deposit and clearing organizations, and stock exchanges) via its SWIFT Transport Network (STN). The types of events processed by SWIFT include:

- **Payments**: Clearing and settlements between member banks.
- **Securities**: Clearing and settlements and cross border electronic trade confirmations.
- **Forex, Money Markets and Derivatives**: Confirmation of trades, marketing and reporting facilities.
- **Trade Finance**: Documenting credits and collections.

The SWIFT ADK e*Way provides secure messaging services (both receiving and transmitting) between SWIFT financial institutions. The SWIFT ADK e*Way is designed specifically to interface with the SWIFTAlliance, and enables the SeeBeyond e*Gate system to exchange data with SWIFTAlliance by providing:

- **Automated integration** of securities events in the new securities standards (events MTxx) which is based on the ISO15022 Data Dictionary. Messages received can be in SWIFT, Telex or Internal formats.
- **Translation** of incoming events received from SWIFT into the format required by existing applications.
- **Security**, by being subject to the same authentication features as other SWIFTAlliance components. Each session between the e*Way and SWIFTAlliance is authenticated using MD5. See **AUTH** on page 94.

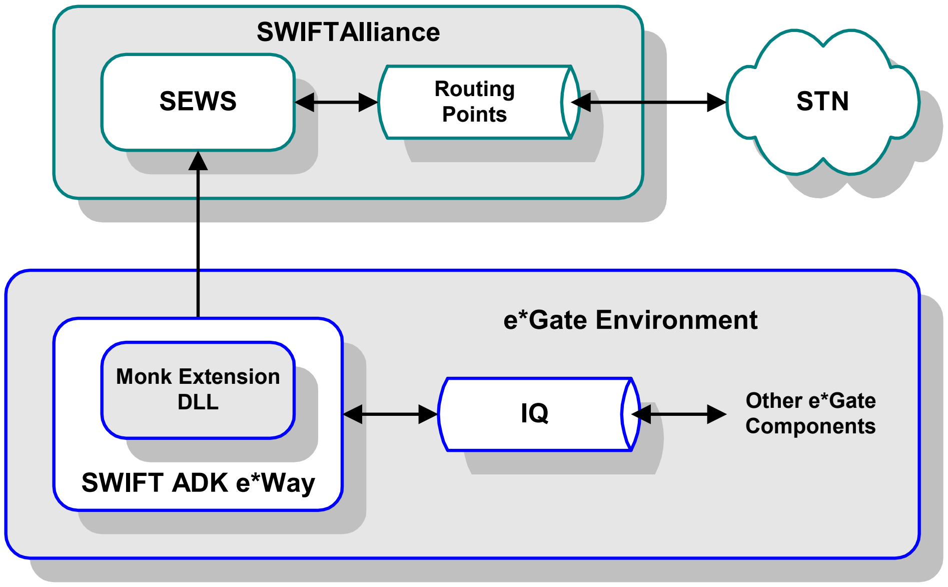
The SWIFT ADK e*Way uses the SWIFT Alliance Developer Toolkit (ADK), which is a library of APIs that can call services provided by SWIFTAlliance servers. For more information about ADK, see the *SWIFT ADK Reference Guide*.

1.2 The SWIFT ADK e*Way

1.2.1 Overview

Within SWIFTAlliance, the SEWS (SWIFT ADK e*Way Service) component is linked to the STN through two routing points, one for incoming Events and the other for outgoing Events. These routing points are analogous to e*Gate Intelligent Queues (IQs).

Figure 1 SWIFT ADK e*Way Overview



Messages received from the SWIFT Network are stored in the routing point. In order to retrieve an Event from the routing point, the ADK component must first reserve the Event and then retrieve it. The Event remains in the routing point until the e*Way sends an acknowledgment to say that it has safely placed the Event in the appropriate IQ. The connection between e*Gate and SWIFT can be monitored in the SWIFTAlliance log.

1.2.2 Components

The SWIFT ADK e*Way includes the following components:

- An executable file (Generic e*Way Kernel), **stcewgenericmonk.exe**
- An accompanying dynamic load library, **stc_swiftadk.dll**, which extends the executable file to form the SWIFT ADK e*Way
- A default configuration file, **SwiftADK.def**
- Monk function scripts and library files, discussed in **Chapter 8**.
- Example schema, discussed in **Sample Schema** on page 35.

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

Note: *The* **SEWS** *ADK component also is provided with the e*Way for installation on the SWIFT Server.*

1.2.3 Availability

The e*Way Intelligent Adapter for SWIFT ADK and the accompanying SEWS Component are available on the following operating systems:

- Windows 2000 SP 1 and Windows 2000 SP 2
- Solaris 8
- AIX 5.1

Chapter 2

Installation

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

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

2.1 System Requirements

To use the e*Way Intelligent Adapter for SWIFT ADK, you need the following:

1. An e*Gate Participating Host, version 4.5.1 or later.
2. A TCP/IP network connection to SWIFTAlliance.
3. Sufficient free disk space to accommodate e*Way files:
   - Approximately 200 KB on Windows systems
   - Approximately 820 KB on Solaris system
   - Approximately 500 KB on AIX systems
4. Additional free disk space on the SWIFTAlliance host for the SEWS component (see additional information under **External System Requirements** on page 15):
   - Approximately 95 KB of disk space on Windows systems
   - Approximately 1.7 MB of disk space on Solaris systems.
   - Approximately 780 KB of disk space 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.*

Environment Configuration

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

2.2 External System Requirements

- The SWIFT ADK e*Way requires a SWIFT ADK runtime license, and supports SWIFTAlliance 5.0.

Note: *The ADK (Alliance Developer Kit) API protocol is supported only by the SWIFTAlliance Access product family. It is* ***not*** *supported by SWIFTAlliance Entry. Customers using SWIFTAlliance Entry can send and receive SWIFT messages through the SeeBeyond Batch e*Way, by appropriately configuring the AFT (Automated File Transfer) interface in SWIFTAlliance Entry.*

2.2.1 External Configuration Requirements

- The SEWS component must be installed and configured (see **Installing SEWS** on page 20 and **Configuring SEWS** on page 53)

Note: *The SEWS component must be installed on the same platform as the SWIFTAlliance server.*

- Before installing SEWS into SWIFTAlliance on UNIX, the root user must set up the correct environment (see installation step 2 under **UNIX Systems** on page 22)

Two routing points, **SEWS_to_egate** and **SEWS_from_egate** are installed when the **SEWS** ADK component is installed. The ways in which these routing points are used, and messages are routed to and from them, are independent of SEWS and depend on the application being used. For information on how to configure routing points, see the SWIFT *System Management Guide*.

2.3 Installing the e*Way

2.3.1 Windows Systems

Installation Procedure

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

To Install the e*Way on a Microsoft Windows System

1 Log in as an Administrator on the workstation on which you want to install the e*Way (*you must have Administrator privileges to install this e*Way*).

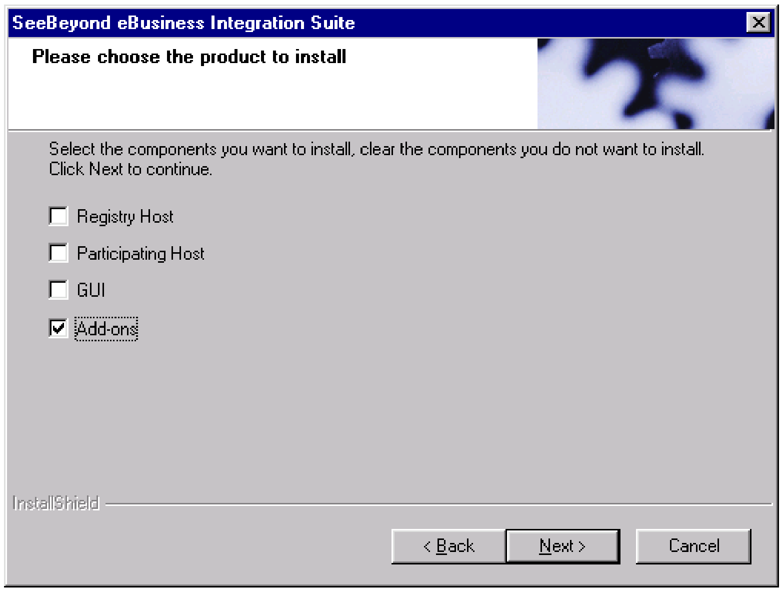
2 Exit all Windows programs and disable any anti-virus applications before running the setup program.

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

4 Launch the setup program.

   A If the CD-ROM drive's Autorun feature is enabled, the setup program should launch automatically. Follow the on-screen instructions until the **Choose Product** dialog box appears (see Figure 2). Check **Add-ons**, then click **Next**.

Figure 2 Choose Product Dialog


SeeBeyond eBusiness Integration Suite

Please choose the product to install

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

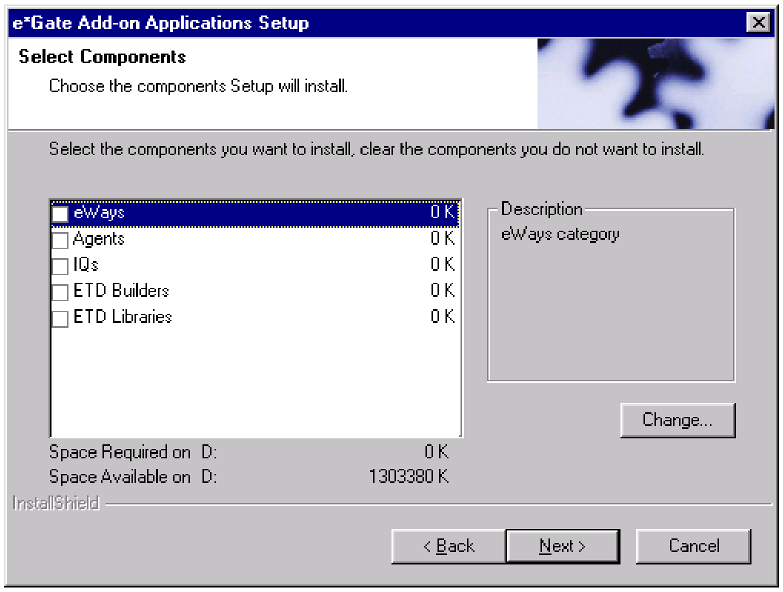
☐ Registry Host
☐ Participating Host
☐ GUI
☑ Add-ons

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

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

5 Follow the on-screen instructions until the **Select Components** dialog box appears (see Figure 3). Highlight—*but do not check*—**eWays** and then click **Change**.

Figure 3 Select Components Dialog



e*Gate Add-on Applications Setup

Select Components

Choose the components Setup will install.

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

☐ eWays	0 K
☐ Agents	0 K
☐ IQs	0 K
☐ ETD Builders	0 K
☐ ETD Libraries	0 K

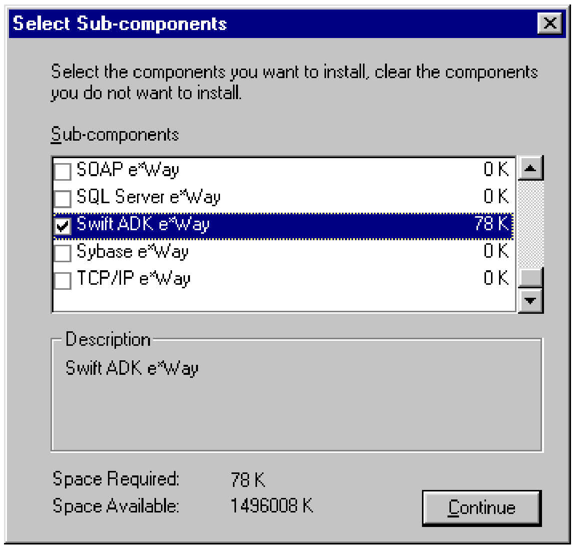
Description
eWays category

Change...

Space Required on D: 0 K
Space Available on D: 1303380 K

6 When the **Select Sub-components** dialog box appears (see Figure 4), check the **Swift ADK e*Way**.

Figure 4 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

☐ SOAP e*Way	0 K
☐ SQL Server e*Way	0 K
✓ Swift ADK e*Way	78 K
☐ Sybase e*Way	0 K
☐ TCP/IP e*Way	0 K

Description
Swift ADK e*Way

Space Required: 78 K
Space Available: 1496008 K

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

8 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_swiftadk.dll
\configs\stcewgenericmonk\	SwiftADK.def
\monk_library\swiftadk\	adk-ack.monk adk-connect.monk adk-disconnect.monk adk-incoming.monk adk-init.monk adk-nak.monk adk-outgoing.monk adk-shutdown.monk adk-startup.monk adk-verify.monk
\monk_library\swiftadk\init\	adk-init-inbond.monk adk-init-outbound.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
\	stcewswiftadk.ctl

2.3.2 UNIX Systems

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

Installation Procedure

1. Log in as root on the workstation containing the CD-ROM drive and, if necessary, mount the CD-ROM drive.
2. Insert the CD-ROM into the drive.
3. At the shell prompt, type

   cd /cdrom
4. Start the installation script by typing:

./setup.sh

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

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/	stcewgenericmonk stc_swiftadk.dll
/configs/stcewgenericmonk/	SwiftADK.def
/monk_library/swiftadk/	adk-ack.monk adk-connect.monk adk-disconnect.monk adk-incoming.monk adk-init.monk adk-nak.monk adk-outgoing.monk adk-shutdown.monk adk-startup.monk adk-verify.monk
/monk_library/swiftadk/init/	adk-init-inbond.monk adk-init-outbound.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
/	stcewswiftadk.ctl

2.4 Installing SEWS

- The SEWS component *must* be installed on the same platform as the SWIFTAlliance server.
- The SWIFTAlliance servers *must not* be running.
- If the Addons installer starts up automatically, select **Cancel**.

2.4.1 Windows Systems

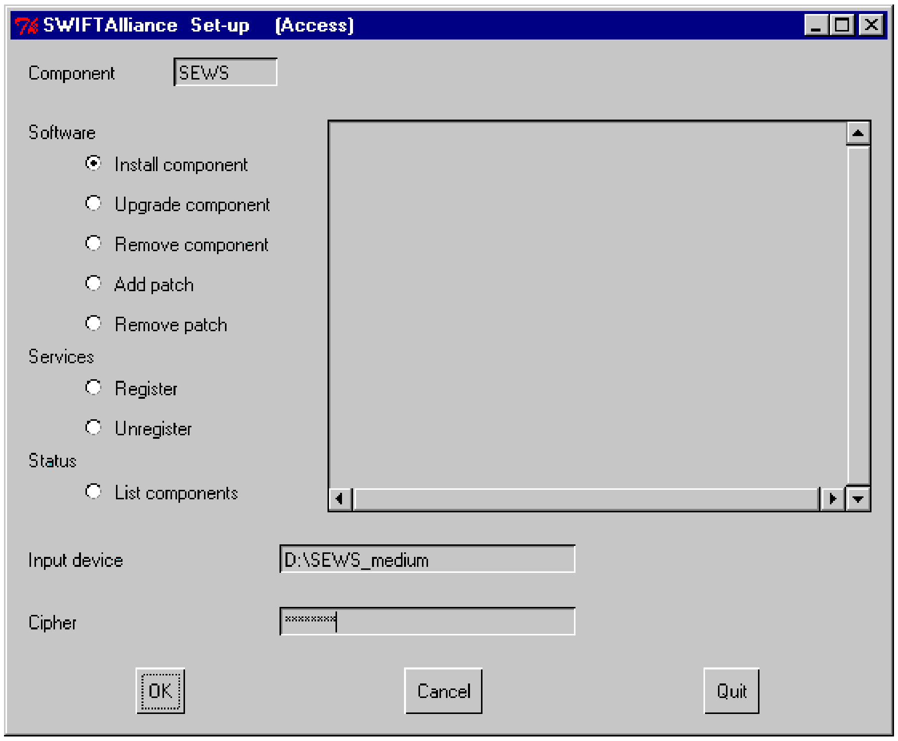
To install SEWS on a Windows 2000 or NT system

1. Unzip the following file to a local directory:

```
<cd>:\SETUP\ADDONS\EWSWIFTADK\SEWS\WIN32\SEWS_MEDIUM.ZIP
```

2. Log in to SWIFTAlliance as the user account under which it was installed.
3. Start the SWIFTAlliance ADK setup program by locating and running the file **ADK_install.exe**. The location of this file depends upon where SWIFTAlliance was installed on your system. The SWIFTAlliance Set-up dialog is then displayed.

Figure 5 SWIFTAlliance Set-up (Install component)



Component: SEWS
Software: Install component [✓]
Input device: D:\SEWS_medium
Cipher: ********

4. In the **Set-up** dialog, install SEWS as follows:
   - A In the **Component** field, type **SEWS**.
   - B In the **Software** group, select the **Install component** option (or **Upgrade component**, if a previous version of SEWS has been installed).

C In the **Input device** field, type in the path to the folder extracted in step 1:

```
<drive>\<temp directory>
```

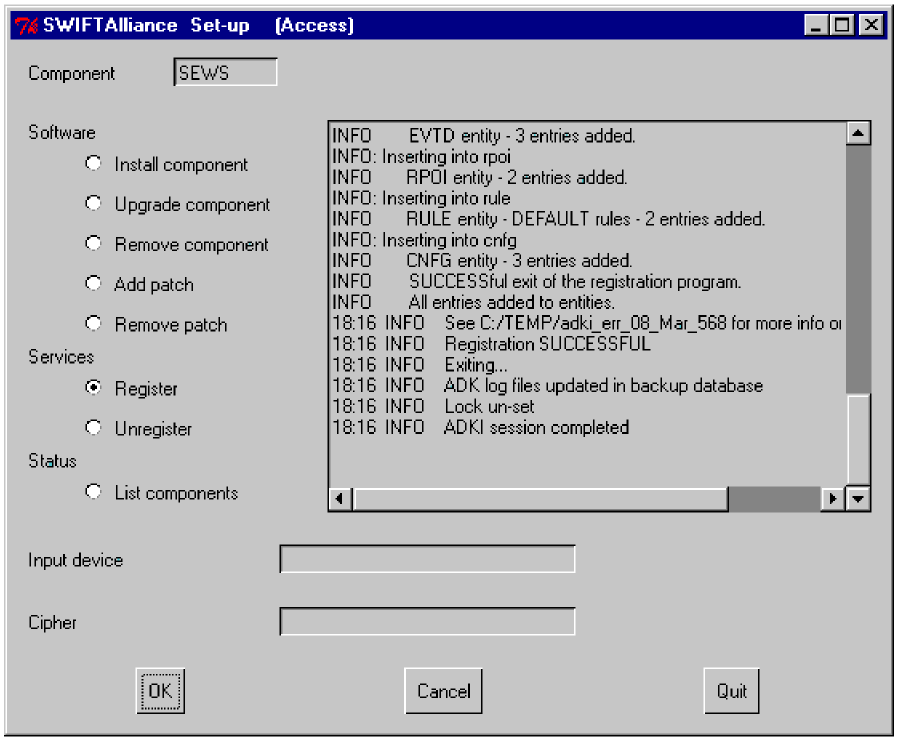
D In the **Cipher** field, type the password provided in:

```
<cd>:\SETUP\ADDONS\EWSWIFTADK\SEWS\README.TXT
```

E Click **OK**. Progress messages are displayed in the message area of the dialog. When you see the following message, SEWS has been installed successfully:

```
ADKI session completed
```

Figure 6 SWIFTAlliance Set-up (Register)



Component: SEWS

Services: Register [✓]

```
INFO      EVTD entity - 3 entries added.
INFO: Inserting into rpoi
INFO      RPOI entity - 2 entries added.
INFO: Inserting into rule
INFO      RULE entity - DEFAULT rules - 2 entries added.
INFO: Inserting into cnfg
INFO      CNFG entity - 3 entries added.
INFO      SUCCESSful exit of the registration program.
INFO      All entries added to entities.
18:16  INFO    See C:/TEMP/adki_err_08_Mar_568 for more info o
18:16  INFO    Registration SUCCESSFUL
18:16  INFO    Exiting...
18:16  INFO    ADK log files updated in backup database
18:16  INFO    Lock un-set
18:16  INFO    ADKI session completed
```

5 If you are performing a first-time installation, register SEWS as follows:

A In the **Set-up** dialog **Software** group, clear the **Install component** option.

B In the **Services** group, select **Register**.

C Clear the **Input device** and **Cipher** fields.

D Click **OK**. Progress messages are displayed in the message area of the dialog. When you see the following message, SEWS has been registered successfully:

```
ADKI session completed
```

E Click **Quit** to close the **Set-up** dialog.

2.4.2 UNIX Systems

To install SEWS on a UNIX system

1. Log in as **root**, running under **/bin/ksh**.
2. Set up the correct environment by sourcing the following script (including the period and space at the beginning):

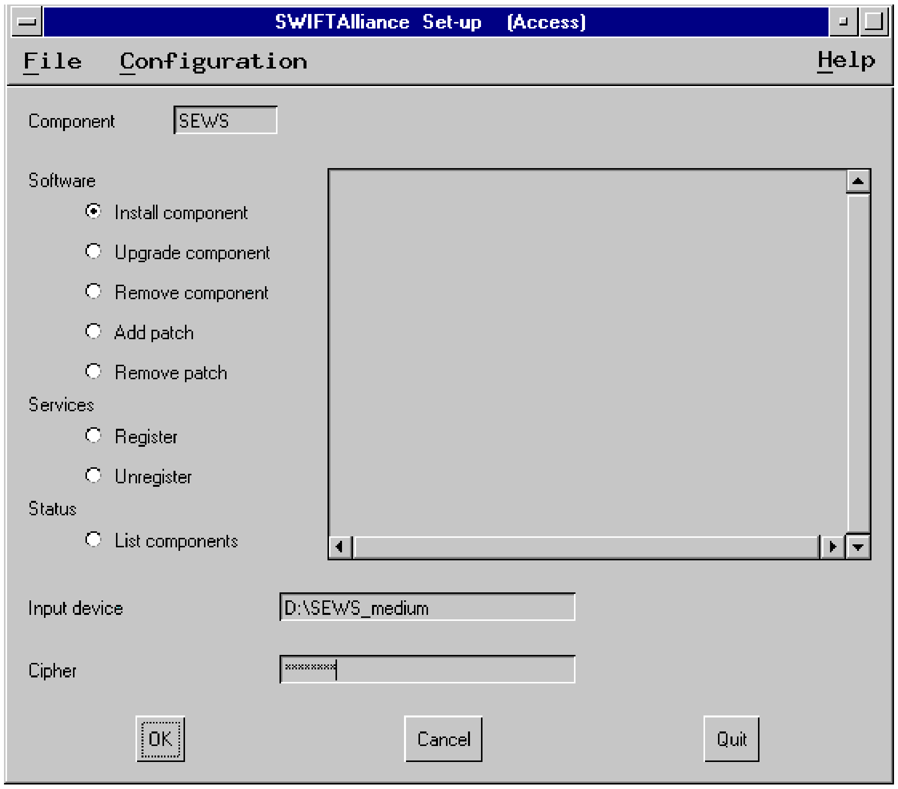
   ```
   . /usr/swa/alliance_init -s
   ```
3. Start the SWIFTAlliance ADK setup program by typing the following command:

   ```
   $(ALLIANCE)/INA/bin/$(ARCH)/adk_install:
   ```

Note: *On Solaris,* **$(ARCH)** *is set to* **'SunOS'**.

The SWIFTAlliance Set-up dialog is then displayed.

Figure 7 SWIFTAlliance Set-up (Install component)



Component: SEWS

Software
- (●) Install component
- ( ) Upgrade component
- ( ) Remove component
- ( ) Add patch
- ( ) Remove patch

Services
- ( ) Register
- ( ) Unregister

Status
- ( ) List components

Input device: D:\SEWS_medium

Cipher: ********

4. In the **Set-up** dialog, install SEWS as follows:

   A. In the **Component** field, type **SEWS**.

   B. In the **Software** group, select the **Install component** option (or **Upgrade component**, if a previous version of SEWS has been installed).

   C. In the **Input device** field, type in the fully-qualified path on the installation CD-ROM:

   ```
   <cd>:\setup\addons\ewswiftadk\sews\sparc8\sews.medium
   ```

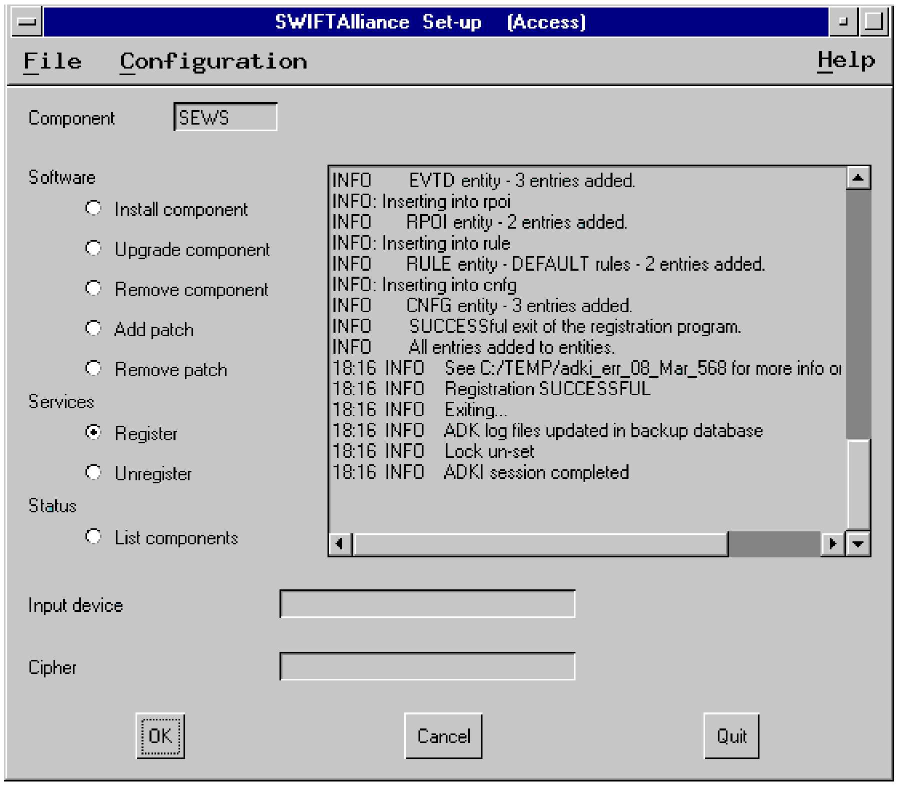
D In the **Cipher** field, type the password provided in:

```
<cd>:\setup\addons\ewswiftadk\sews\readme.txt
```

E Click **OK**. Progress messages are displayed in the message area of the dialog. When you see the following message, SEWS has been installed successfully:

```
ADKI session completed
```

Figure 8 SWIFTAlliance Set-up (Register)



Component: SEWS

Services: [✓ Register]

```
INFO      EVTD entity - 3 entries added.
INFO: Inserting into rpoi
INFO      RPOI entity - 2 entries added.
INFO: Inserting into rule
INFO      RULE entity - DEFAULT rules - 2 entries added.
INFO: Inserting into cnfg
INFO      CNFG entity - 3 entries added.
INFO      SUCCESSful exit of the registration program.
INFO      All entries added to entities.
18:16 INFO   See C:/TEMP/adki_err_08_Mar_568 for more info o
18:16 INFO   Registration SUCCESSFUL
18:16 INFO   Exiting...
18:16 INFO   ADK log files updated in backup database
18:16 INFO   Lock un-set
18:16 INFO   ADKI session completed
```

5 If you are performing a first-time installation, register SEWS as follows:

A In the **Set-up** dialog **Software** group, clear the **Install component** option.

B In the **Services** group, select **Register**.

C Clear the **Input device** and **Cipher** fields.

D Click **OK**. Progress messages are displayed in the message area of the dialog. When you see the following message, SEWS has been registered successfully:

```
ADKI session completed
```

E Click **Quit** to close the **Set-up** dialog.

2.5 Optional Example Files

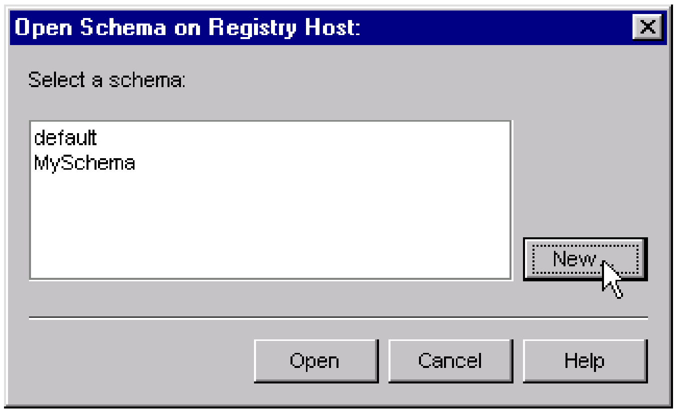
The installation CD-ROM contains a sample schema, **adk_sample**, located in the **samples\ewswiftadk** directory. To use a schema, you must load it onto your system using the following procedure. See **Sample Schema** on page 35 for descriptions of the sample schema and instructions regarding its use.

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

2.5.1 Installation Procedure

1 Invoke the **Open Schema** dialog box and select **New** (see Figure 9).

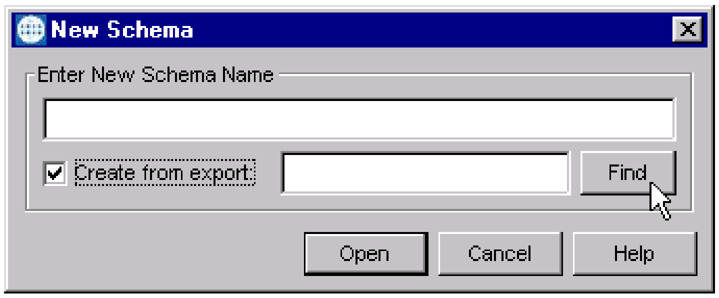
Figure 9 Open Schema Dialog



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

3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 10).

Figure 10 New Schema Dialog



4 Navigate to **adk_sample.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* **adk_sample.zip**

to a local directory and extract the files. Using a text editor, edit the file **adk_sample.exp**, *replacing all instances of the name* **localhost** *with your desired name. Add the edited* **.exp** *file back into the* **.zip** *file.*

2.5.2 Subdirectories and Files

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

Table 5 Subdirectories and Files Installed by Sample Schema Load

Subdirectories	Files
\	adk_sample.ctl
\runtime\configs\stcewfile\	adk_sample_input.cfg adk_sample_input.sc adk_sample_output.cfg adk_sample_output.sc
\runtime\configs\stcewgenericmonk\	adk_sample_eway.cfg adk_sample_eway.sc

Chapter 3

System Implementation

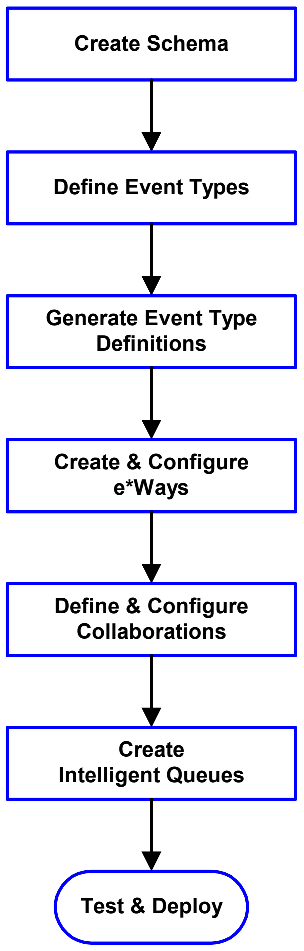
This chapter describes the procedure for implementing your system, incorporating the SWIFTAlliance, SEWS, and the SWIFT ADK e*Way.

3.1 Overview

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

The following topics are discussed in this chapter:

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 28).

2 The second step is to define the Event Types you are transporting and processing within the Schema (see **Creating Event Types** on page 29).

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 29).

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 31).

6 Now you need to create Intelligent Queues to hold published Events (see **Creating Intelligent Queues** on page 32

7 Finally, you must test your Schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

3.1.2 Viewing e*Gate Components

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

3.2 Creating a Schema

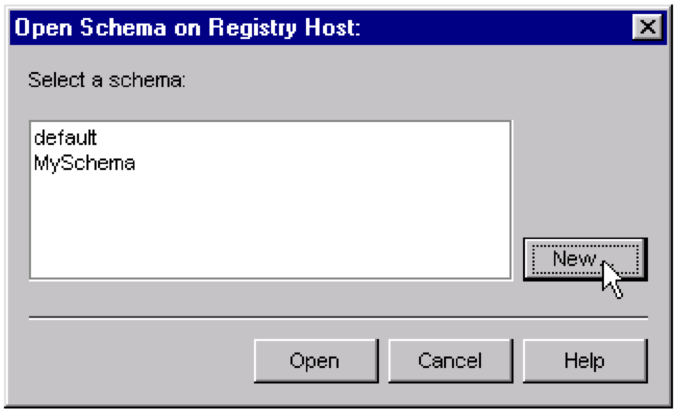
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

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

To select or create a schema

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

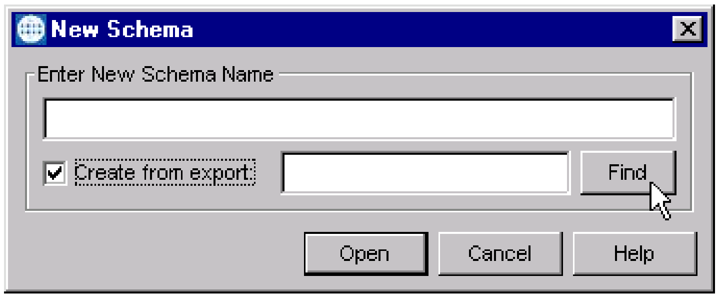
Figure 11 Open Schema Dialog



default
MySchema

2 Clicking **New** invokes the **New Schema** dialog box (Figure 12).

Figure 12 New Schema Dialog



3 Enter a new schema name and click **Open**.

4 The e*Gate Enterprise Manager then opens under your new schema name.

5 From the **Options** menu, click on **Default Editor** and select **Monk**.

6 Select the **Components** tab, found at the bottom of the Navigator pane of the e*Gate Enterprise Manager window.

7 You are now ready to begin creating the necessary components for this new schema.

3.3 Creating Event Types

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

To define the Event Types

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

3.4 Creating Event Type Definitions

Before e*Gate can process any data, you must create an Event Type Definition to package and route that data within the e*Gate system. In the case of SWIFT, the SeeBeyond SWIFT ETD Library provides pre-defined templates for the full range of SWIFT data types. See the *SWIFT ETD Library User's Guide* for more information.

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

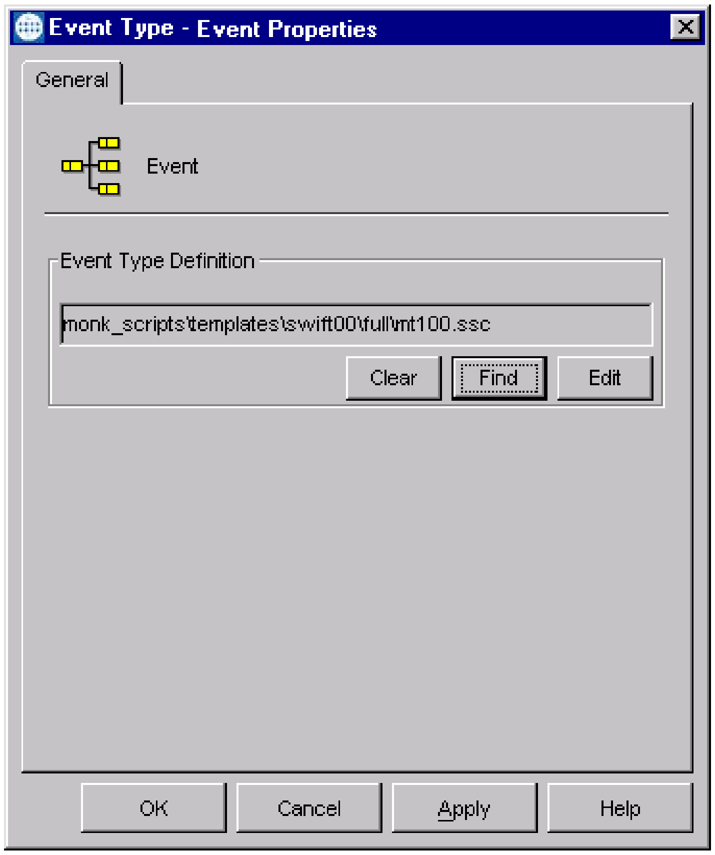
3.5 Assigning ETDs to Event Types

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 13.

Figure 13 Event Type Properties Dialog Box



monk_scripts\templates\swift00\full\mt100.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\templates\swift<yy>\<full or slim>** folder, then select the desired file name (**mt<nnn>.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.

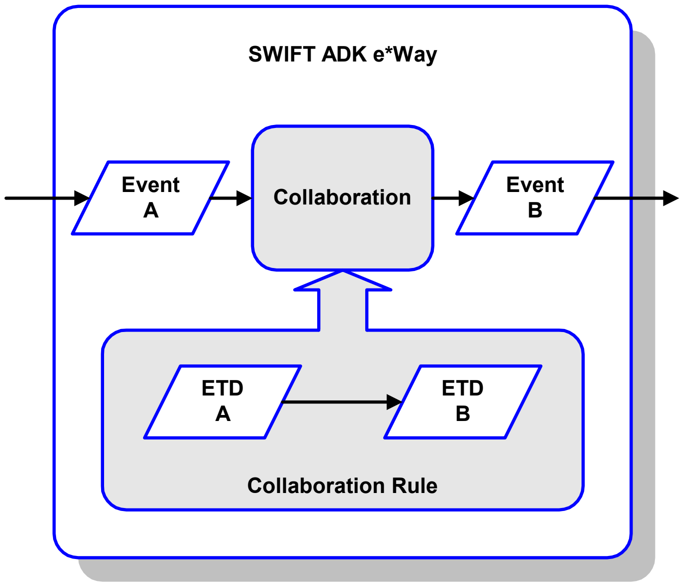
Each Event Type is associated with the specified Event Type Definition.

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 14 Collaborations



The Collaboration is driven by a Collaboration Rules script, which defines the relationship between the incoming and outgoing ETDs. You can use an existing Collaboration Rules script, or use the Monk programming language to write a new Collaboration Rules 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 **.ssc**.

3.7 Creating Intelligent Queues

IQs are components that provide nonvolatile storage for Events within the e*Gate system as they pass from one component to another. IQs are *intelligent* in that they are more than just a "holding tank" for Events. They actively record information about the current state of Events.

Each schema must have an IQ Manager before you can add any IQs to it. You must create at least one IQ per schema for published Events within the e*Gate system. Note that e*Ways that publish Events externally do not need IQs.

For more information on how to add and configure IQs and IQ Managers, see the *e*Gate Integrator System Administration and Operations Guide.* See the *e*Gate Integrator Intelligent Queue Services Reference Guide* and the *SeeBeyond JMS Intelligent Queue User's Guide* for complete information on working with IQs.

3.8 Troubleshooting

Startup information, shutdown conditions, and fatal errors can be examined in the SWIFTAlliance journal log.

3.8.1 Environmental Variables

Occasionally '**/usr/swa/alliance_init**' does not set **$ALLIANCE** or **$ARCH**. If this occurs, use '**/usr/swa/alliance_init -S >init.out**' and then run **init.out** with '**. ./init.out**'.

On Solaris, **$ARCH** is set to '**SunOS**'.

3.8.2 Error Codes

The following error codes can be returned:

Code	Content	Meaning
201	"No messages to get"	The requested message does not exist.
501	"Unknown state"	The state given in the request was not recognized.
502	"No s_umid supplied"	The ACK/NAK request cannot succeed without a s_umid.
503	"No s_umid to ACK"	There is no outstanding message requiring acknowledgment.
504	"No s_umid to NAK"	There is no outstanding message requiring acknowledgment.

Code	Content	Meaning
505	"Incorrect s_umid, pending s_umid supplied"	The s_umid given in the request is not the s_umid requiring acknowledgment. This response contains two additional arguments. ▪ s_umid: The s_umid of the message that is pending acknowledgment. ▪ instance: The instance number of that message.
506	"Failed to route message: ..."	After the message was added to the routing point, it couldn't be "routed on" in SWIFT terminology. The reason for this failure is included in the response Content.
507	"Failed to reserve message: ..."	The message specified could not be reserved. This response contains additional arguments on why the request failed. ▪ s_umid: The s_umid of the message that could not be acknowledged. ▪ instance: The instance number of the message that could not be acknowledged.
509	"Failed to get message: ..."	SEWS was not able to retrieve the message. The response content provides text that explains the reason for the error.
510	"Failed to get message, then failed to unreserve it: ..."	SEWS reserved the message, but could not retrieve it, and then could not unreserve the same message when recovering. Includes text further detailing the problem in the content.
511	"Failed to count/list instances ..."	SWIFTAlliance couldn't perform the operation. The human readable text - Content - includes the ADK error string that details the problem.

Code	Content	Meaning
513	"Failed to add message: ..."	The message could not be added to the SWIFT routing point. The response content provides additional detail on why the procedure failed. When this error occurs, there are two additional arguments in the response. ▪ offset: The character offset into the text message, that caused the error. ▪ reason: Text giving further details about the error.
514	"Unknown message type"	The type argument contained an unexpected value. The response also contains a type argument: ▪ type: The type value given in the initial request that was not understood.
515	"Message type not supplied"	No type argument was supplied with the initial GET request.
516	"Pending ACK for another message"	No more messages can be retrieved until the last message is acknowledged. Responses with this code include another argument "pending", which lists the s_umid of the pending message.
517	"Invalid routing point name supplied"	The routing point name given in the request was not recognized.
518	"No authorization"	Authentication was not possible.
519	"Too many clients connected"	SEWS cannot take more than one client per direction.
550	"Cannot translate SWIFT message to ADK"	The SWIFT message supplied in the request cannot be understood by SEWS. Perhaps there is a Signature error in the message. More exact details can be found in the SWIFTAlliance logs.
551	"Cannot translate TELEX message to ADK"	The Telex message supplied in the request cannot be understood by SEWS. Perhaps there is a Signature error in the message. More exact details can be found in the SWIFTAlliance logs.

3.9 Sample Schema

A bidirectional *sample schema* is included with the e*Way, which can be used to establish connection and authentication using the SEWS component. To use the sample schema:

1 Install the sample schema using the e*Gate Schema Import facility (see **Optional Example Files** on page 24.

3.9.1 adk_sample

This sample schema (see **Figure 15 on page 36**)sets up one SWIFT ADK e*Way and two File e*Ways having the logical names shown in the following table.

e*Way Type	Logical Name
SWIFT ADK e*Way	adk_sample_eway
File e*Way	adk_sample_output adk_sample_input

It also sets up two Intelligent Queues (IQs), with the logical names **IQ1** and **IQ2**. The File e*Ways substitute for external applications that would exist in a production environment.

Note that **adk_sample_input** requires the following configuration parameter values (see the *Standard e*Way Intelligent Adapter User's Guide* for additional information).

Poller (inbound) Settings Parameter	Value
Remove EOL	no
Multiple Records Per File	no

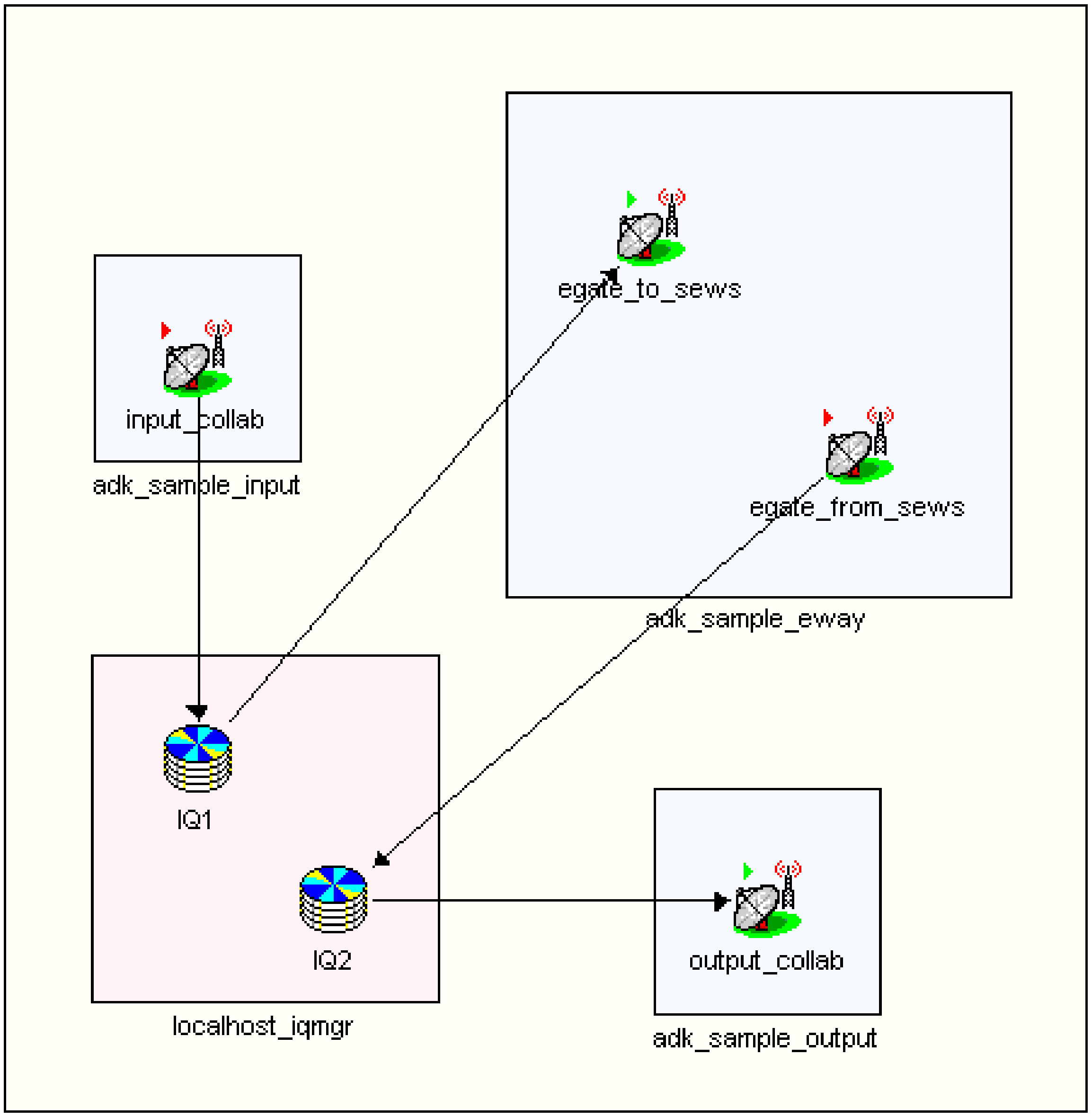
In this example, the SWIFT ADK e*Way, **adk_sample_eway**, establishes a connection with the SWIFT server and authenticates itself using the SEWS component. It requests messages from the configured e*Gate-inbound routing point on the SWIFT server and passes them to the IQ. The File e*Way **adk_sample_output** receives these messages and posts them to a file.

Messages to SWIFT are retrieved from a number of files by **adk_sample_input** and the messages passed to the IQ. The SWIFT ADK e*Way retrieves the messages from the IQ and passes them to the e*Gate-outbound routing point on the SWIFT server using SEWS.

In the example monk scripts, error control has been implemented within the **adk-connect** function. When a connection to the SWIFT server is established and authorized, the e*Way checks for reserved messages on both routing points. If any message is left in a reserved state on either of the routing points (**SEWS_to_egate**, or **SEWS_from_egate**), then:

- The reserved message is routed automatically
- The next message is marked as being a potential duplicate

Figure 15 adk_sample Schema



e*Gate to SEWS

1. Messages to be sent to SWIFT are contained in files having the extension **.fin**. These are retrieved by **adk_sample_input** as Event Type **GenericInEvent**.
2. The messages are published to **IQ1** as Event Type **GenericInEvent**. The pass-through Collaboration used, **input_collab** performs a byte-by-byte duplication (see Figure 16).
3. The Collaboration **egate_to_SEWS** subscribes to Events of type **GenericInEvent** in the IQ and publishes Events of type **GenericInEvent** (see Figure 17).
4. The SWIFT ADK e*Way, **adk_sample_eway**, performs the Collaboration and routes the messages to the e*Gate-outbound routing point, **SEWS_from_egate**, on the SWIFT server.

Figure 16 input_collab Collaboration

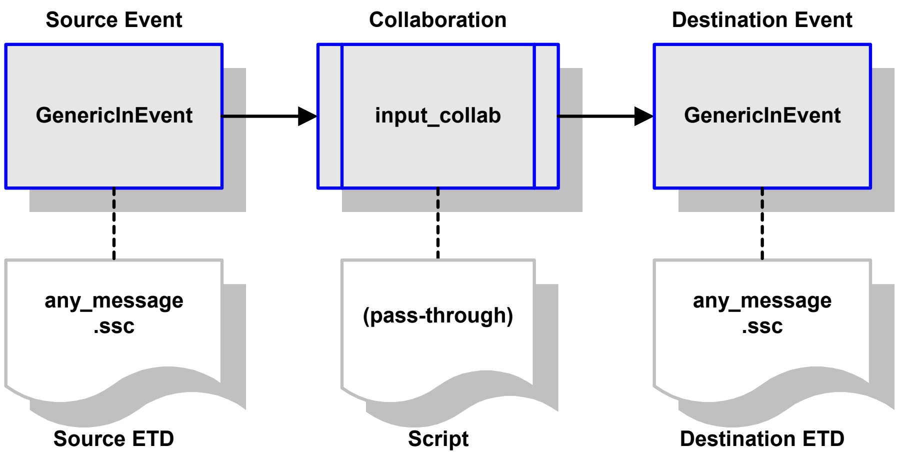
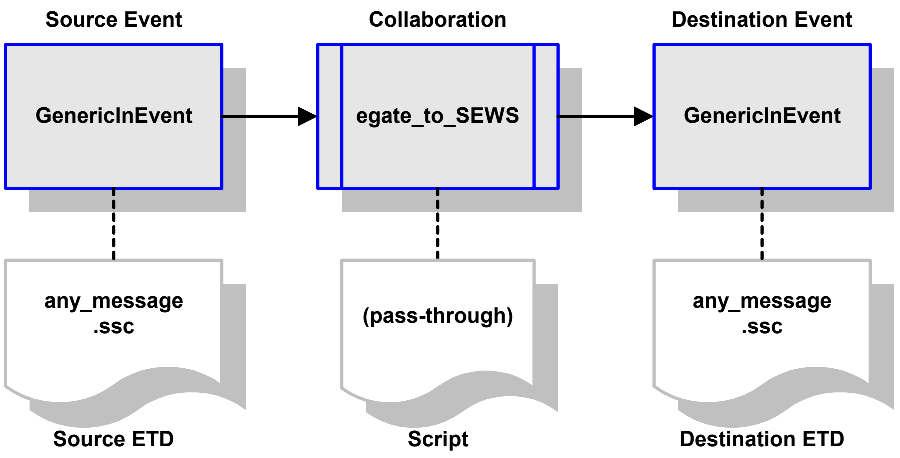


Figure 17 SWIFT ADK e*Way Outbound Collaboration



SEWS to e*Gate

1 Messages are retrieved one at a time from the e*Gate-inbound routing point, **SEWS_to_egate**, by **adk_sample_eway**.

2 The incoming Event Type, **GenericInEvent**, is passed to **IQ2** as Event Type **GenericInEvent**. The pass-through Collaboration **egate_from_SEWS** performs a byte-by-byte duplication (see Figure 18).

3 The Collaboration **output_collab** subscribes to Events of type **GenericInEvent** in the IQ and writes them to the file **swift%d.dat** as Event Type **GenericInEvent** (see Figure 19).

Figure 18 SWIFT ADK e*Way Inbound Collaboration

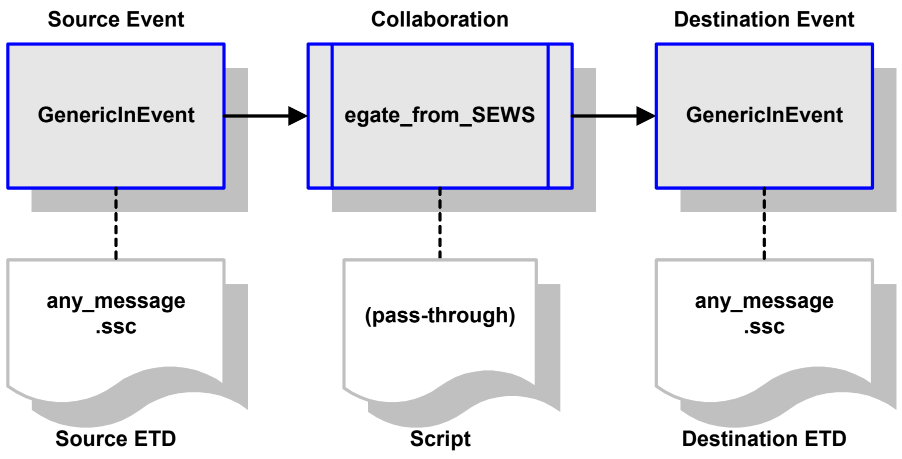
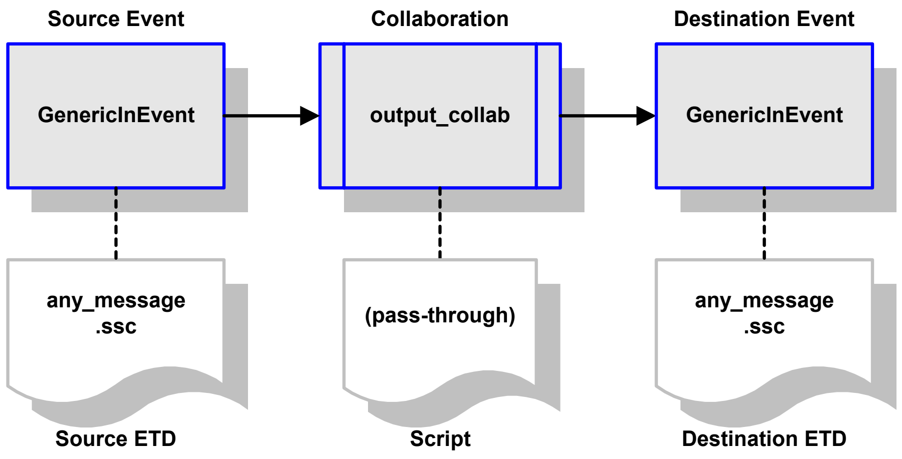


Figure 19 outbound_collab Collaboration



Chapter 4

e*Way Setup

This chapter describes the procedures for customizing the SWIFT ADK e*Way to operate with your system.

4.1 Overview

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

The topics discussed in this chapter include the following:

4.2 Setting Up the e*Way

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

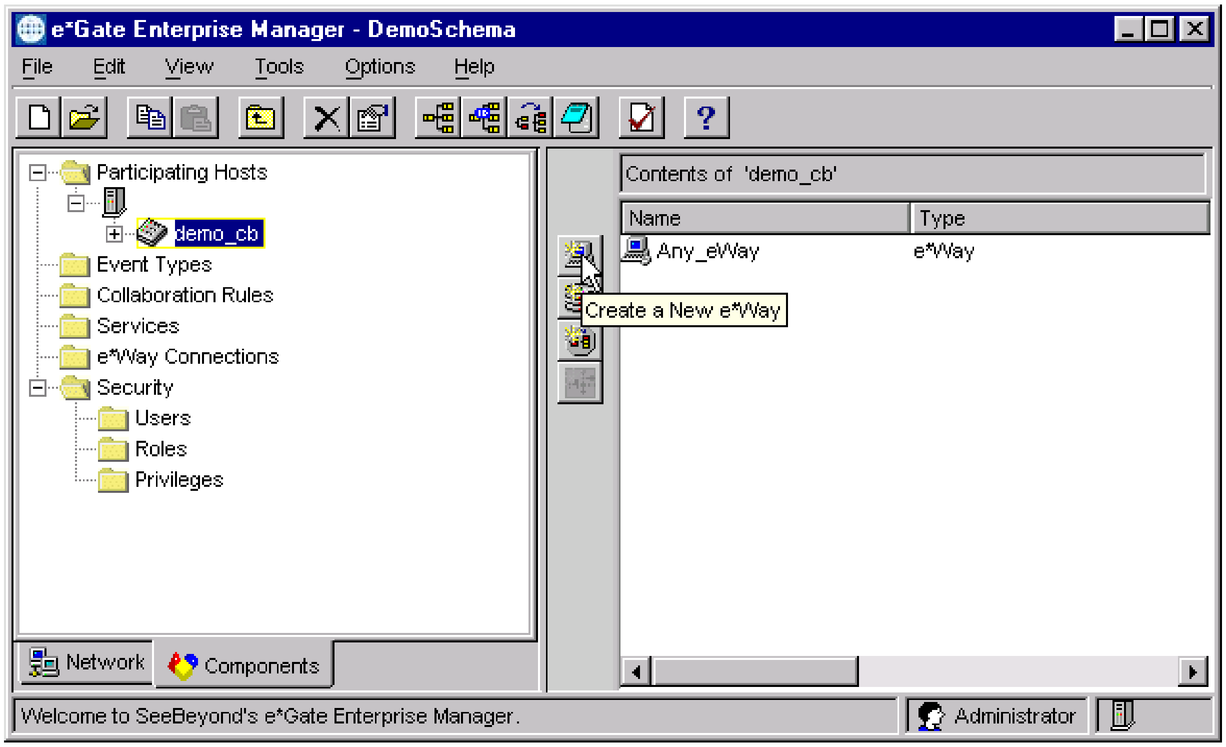
4.2.1 Creating the e*Way

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 20 e*Gate Enterprise Manager Window (Components View)



Welcome to SeeBeyond's e*Gate Enterprise Manager.

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.

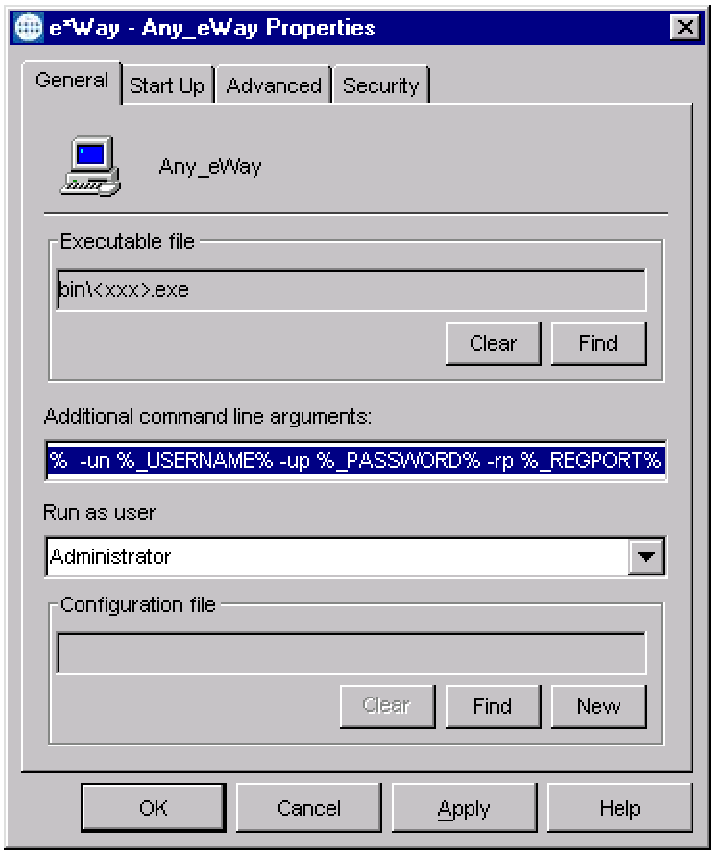
4.2.2 Modifying e*Way Properties

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 21).

Note: *The executable and default configuration files used by this e*Way are listed in* **Components** *on page 13.*

Figure 21 e*Way Properties (General Tab)



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

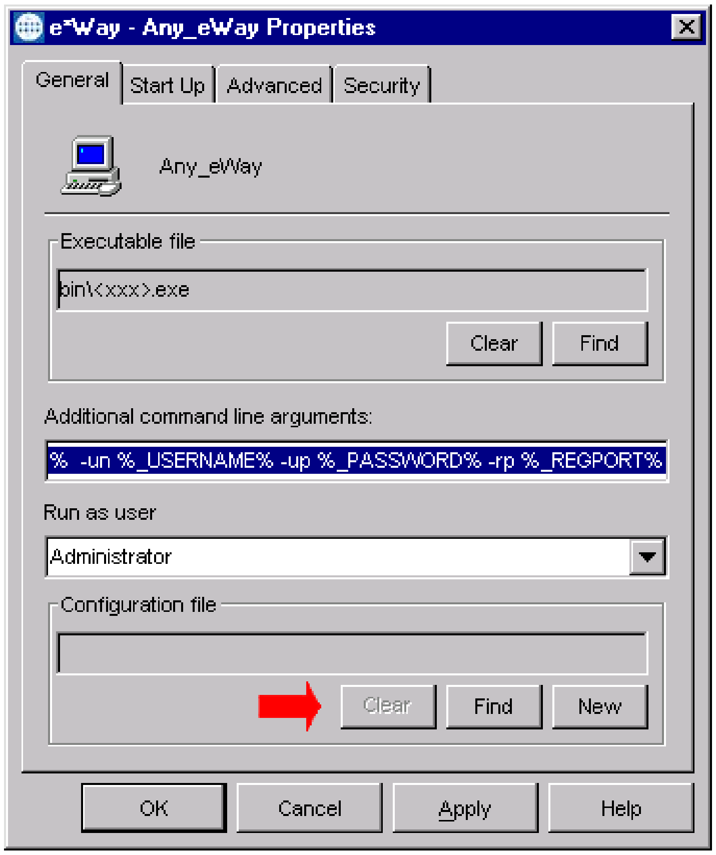
The e*Way's default configuration parameters are stored in an ASCII text file with a **.def** extension. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (**.cfg**) file.

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.
2. The appropriate executable file should be displayed automatically. If not, select the correct executable file by clicking **Find** and navigating to the file.

Note: *The executable and default configuration files used by this e*Way are listed in* **Components** *on page 13.*

Figure 22 e*Way Properties - General Tab


e*Way - Any_eWay Properties

Any_eWay

Executable file: bin\<xxx>.exe

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

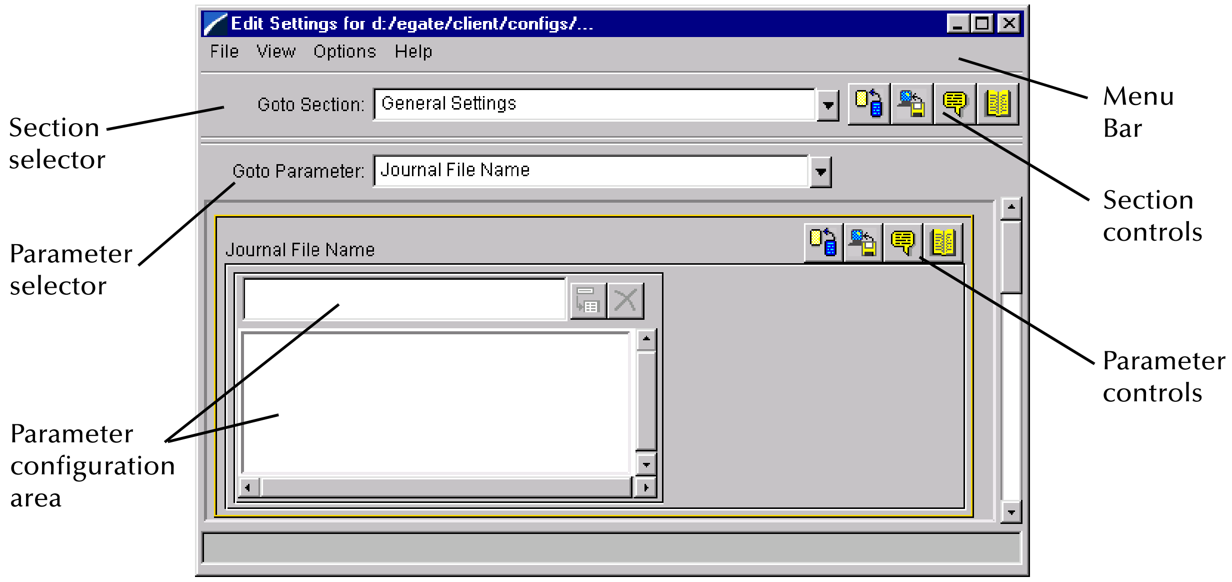
Run as user: Administrator

Configuration file:

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

4.2.4 Using the e*Way Editor

Figure 23 The e*Way Configuration Editor


Edit Settings for d:/egate/client/configs/...
File View Options Help
Goto Section: General Settings
Goto Parameter: Journal File Name
Journal File Name
Section selector
Parameter selector
Parameter configuration area
Menu Bar
Section controls
Parameter controls

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 6 below.

Table 6 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 7

Table 7 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*.

4.2.5 Changing the User Name

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

To change the user name

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

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

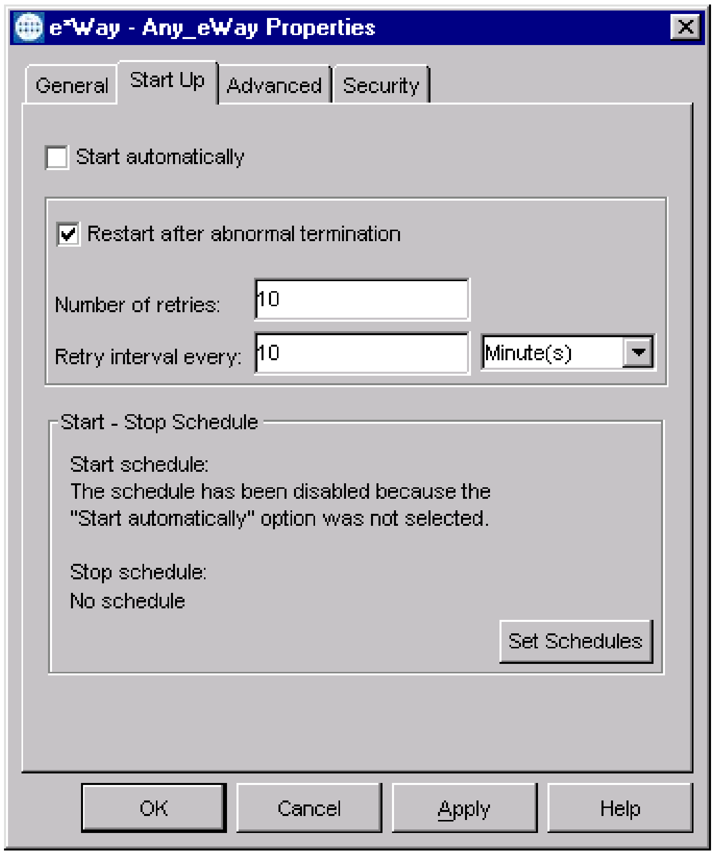
4.2.6 Setting Startup Options or Schedules

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

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

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see Figure 24). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 24 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**.

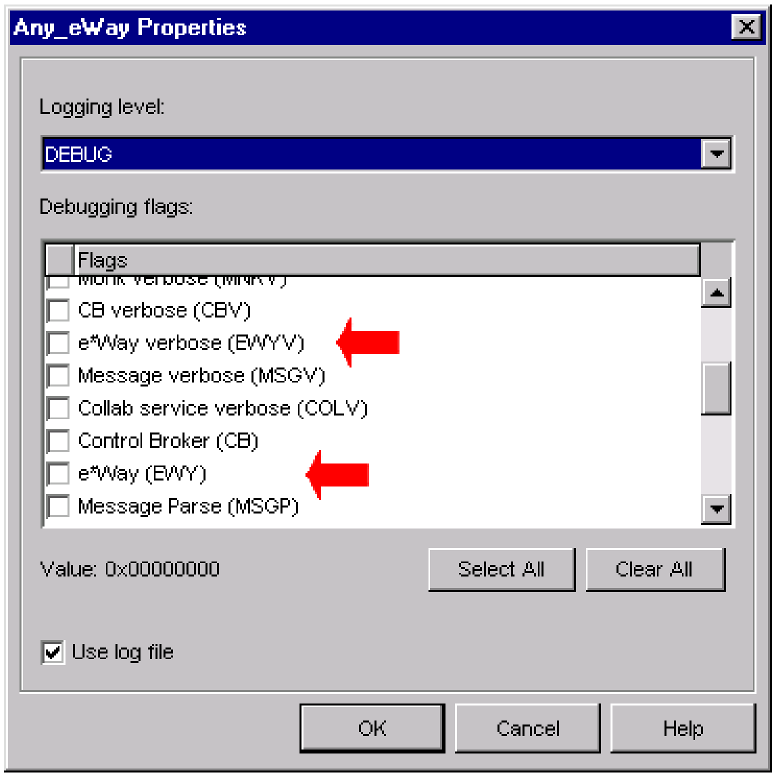
4.2.7 Activating or Modifying Logging Options

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

To set the e*Way debug level and flag

1. Display the e*Way's Properties dialog.
2. Select the **Advanced** tab.
3. Click **Log**. The dialog window appears, as shown in Figure 25.

Figure 25 e*Way Properties (Advanced Tab - Log Option)


Any_eWay Properties

Logging level: DEBUG

Debugging flags:

- [ ] 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 impact on system performance.
6. Click **OK**.

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

4.2.8 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way sends a Monitoring Event to the Control Broker, which 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 "feed" this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e*Way "feeds" properly configured, and are they subscribing to the appropriate Events correctly?

In the e*Way Editor

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

On the e*Way's Participating Host

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

In the SWIFT Application

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

4.3.2 System-related Problems

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

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

4.3.3 Operational Problems

Startup information, shutdown conditions, and fatal errors can be examined in the SWIFTAlliance journal log.

Chapter 5

SEWS Setup

This chapter describes the procedures for customizing the SEWS component to operate with your system.

5.1 Overview

The SEWS component must be configured to properly connect the e*Way with SWIFTAlliance. Note that the test environment listed below is optional, and is included in case you do not have an existing environment appropriate for testing purposes.

The topics discussed in this chapter include the following:

Note: *Although the figures in this chapter show only the Windows user interface, the UNIX interface is nearly identical.*

5.2 Configuring SEWS

To configure SEWS, first set your SWIFTAlliance servers to **housekeeping** mode, then set up the following items to match your system and your schema:

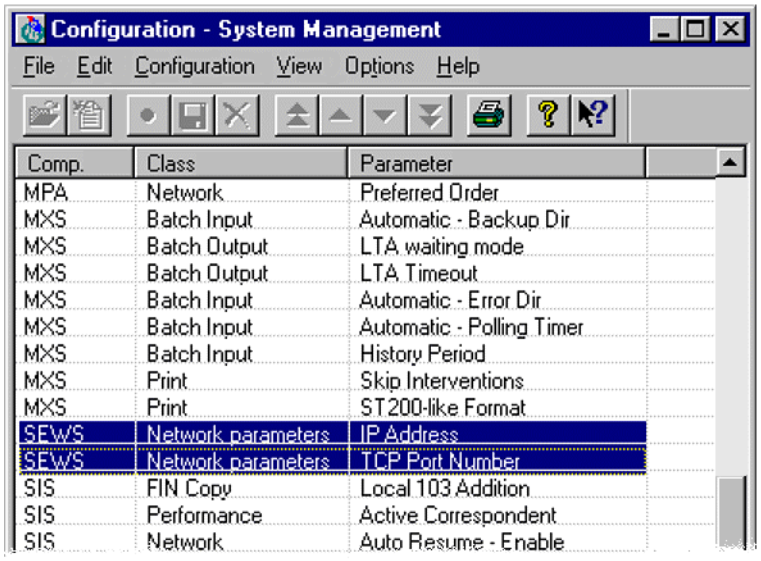
- TCP/IP Parameters
- Security Options

Note: *The values entered for the SEWS parameters must exactly match the corresponding e*Way configuration parameters.*

5.2.1 TCP/IP Parameters

The Systems Management window of SWIFTAlliance allows you to configure the two *non-secret* parameters of the SEWS component: the **IP Address** and **TCP Port Number**, both highlighted in Figure 26.

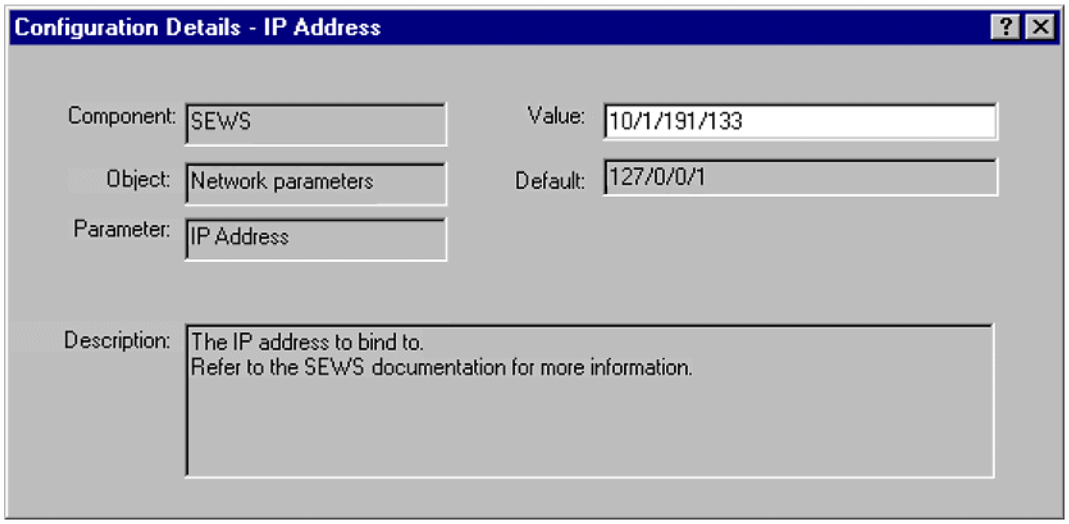
Figure 26 Systems Management Window



Comp.	Class	Parameter
MPA	Network	Preferred Order
MXS	Batch Input	Automatic - Backup Dir
MXS	Batch Output	LTA waiting mode
MXS	Batch Output	LTA Timeout
MXS	Batch Input	Automatic - Error Dir
MXS	Batch Input	Automatic - Polling Timer
MXS	Batch Input	History Period
MXS	Print	Skip Interventions
MXS	Print	ST200-like Format
SEWS	Network parameters	IP Address
SEWS	Network parameters	TCP Port Number
SIS	FIN Copy	Local 103 Addition
SIS	Performance	Active Correspondent
SIS	Network	Auto Resume - Enable

Double-clicking the **IP Address** line presents the dialog box, shown in Figure 27.

Figure 27 Systems Management—IP Address Window


Configuration Details - IP Address

Component:	SEWS	Value:	10/1/191/133
Object:	Network parameters	Default:	127/0/0/1
Parameter:	IP Address		

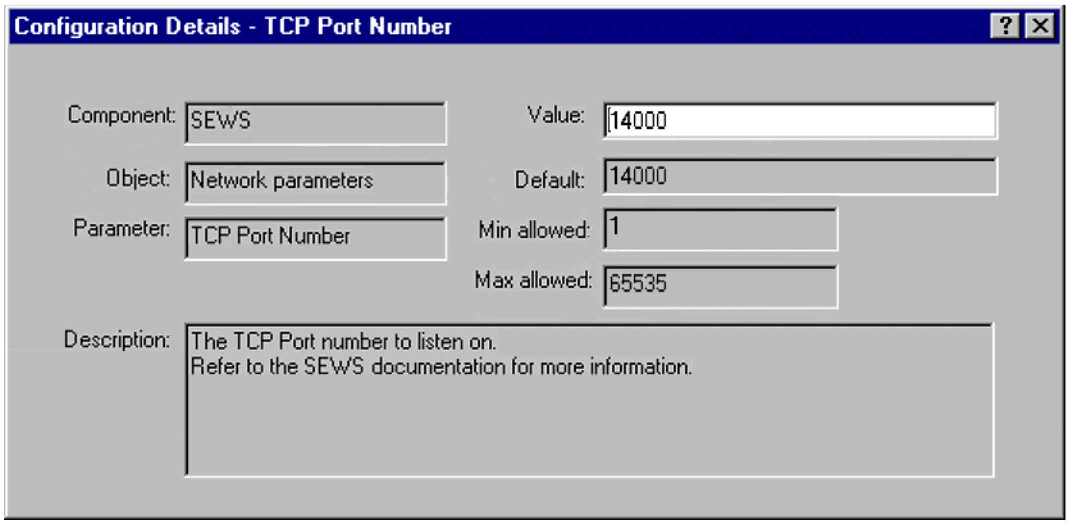
Description: The IP address to bind to.
Refer to the SEWS documentation for more information.

In this example, the value is set to **10.1.191.133**—the dots are replaced with slashes, however, to be compatible with the current ADK software (see following *Note*). Note also that the default value shown (**127/0/0/1**) *does not work*—it is only a placeholder, and *must* be replaced with the IP address of the host running SWIFT Alliance. The value you enter for the **IP Address** parameter must match the value you enter later for the **Hostname** parameter in the e*Way configuration. See **SWIFT Setup** on page 90.

Note: *Currently, dot characters are not reliably stored because of a known bug in ADK. To store an IP address, you must substitute slashes (/) for dots (.) in the entry. For example, the default IP address* **127.0.0.1** *is entered as* **127/0/0/1**.

Double-clicking the **TCP Port Window** line in the Systems Management window presents the dialog box shown in Figure 28.

Figure 28 Systems Management—TCP Port Window


Configuration Details - TCP Port Number

Component:	SEWS	Value:	14000
Object:	Network parameters	Default:	14000
Parameter:	TCP Port Number	Min allowed:	1
		Max allowed:	65535

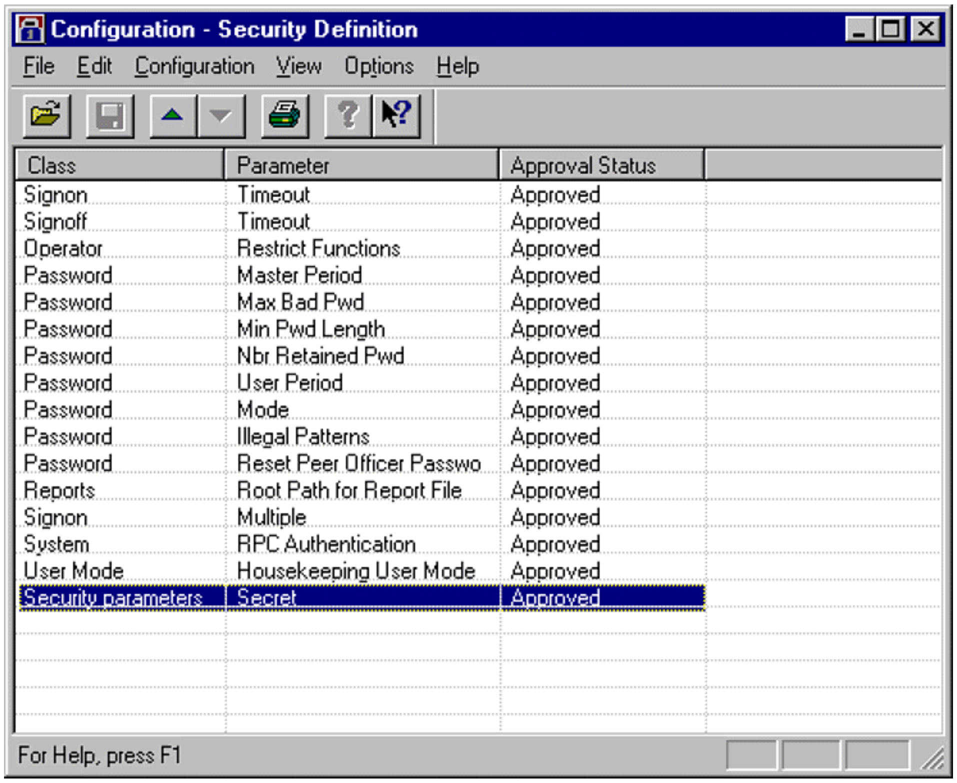
Description: The TCP Port number to listen on.
Refer to the SEWS documentation for more information.

The **TCP Port Number** defaults to **14000**, which is generally safe to leave as-is. The value you enter for this parameter must match the value you enter later for the **Port Number** parameter in the e*Way configuration. See **SWIFT Setup** on page 90.

5.2.2 Security Options

The Security Definition window of SWIFTAlliance allows you to configure the *secret* parameters of the SEWS component. Any changes made to any of the security definitions must be approved by both security officers **LSO** and **RSO** (which stand for Left and Right Security Office, respectively) before they can become effective.

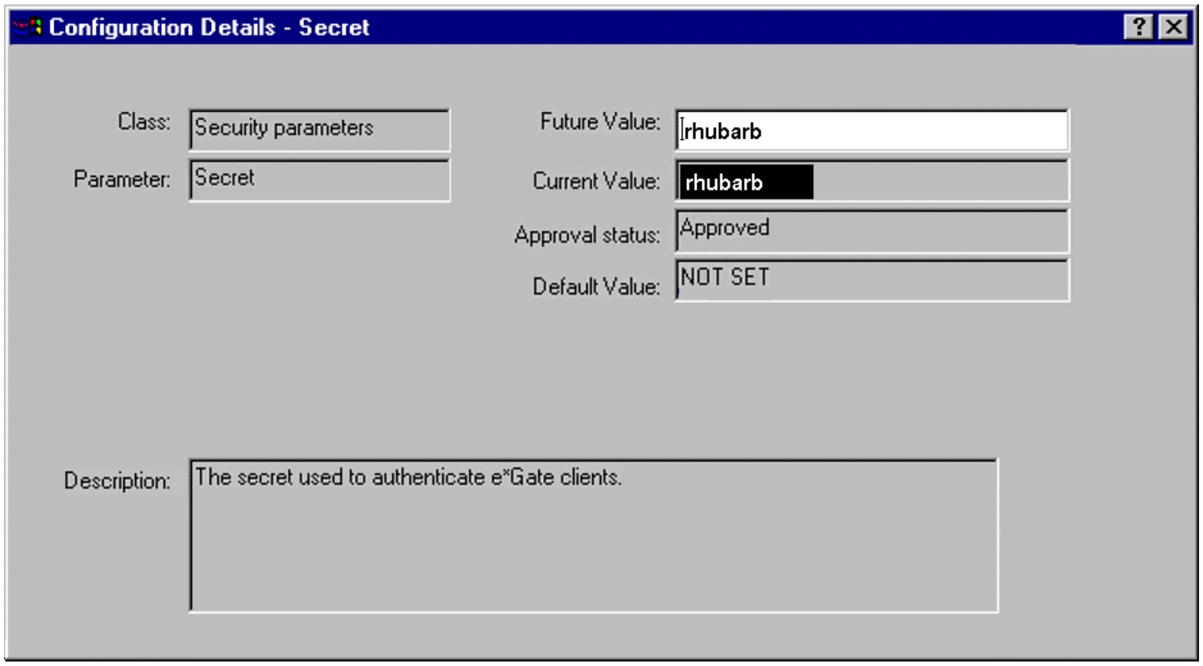
Figure 29 Security Definition Window


Configuration - Security Definition
File Edit Configuration View Options Help
For Help, press F1

Class	Parameter	Approval Status
Signon	Timeout	Approved
Signoff	Timeout	Approved
Operator	Restrict Functions	Approved
Password	Master Period	Approved
Password	Max Bad Pwd	Approved
Password	Min Pwd Length	Approved
Password	Nbr Retained Pwd	Approved
Password	User Period	Approved
Password	Mode	Approved
Password	Illegal Patterns	Approved
Password	Reset Peer Officer Passwo	Approved
Reports	Root Path for Report File	Approved
Signon	Multiple	Approved
System	RPC Authentication	Approved
User Mode	Housekeeping User Mode	Approved
Security parameters	Secret	Approved

In Figure 29, the **Secret** parameter of SEWS is highlighted. The **Secret** parameter is used to authenticate a connection from the e*Way, in the same way that a password is used.

Figure 30 Security Definition—Secret Window



Configuration Details - Secret

Class:	Security parameters	Future Value:	rhubarb
Parameter:	Secret	Current Value:	rhubarb
		Approval status:	Approved
		Default Value:	NOT SET

Description: The secret used to authenticate e*Gate clients.

In Figure 30 we see the **Secret** value, which, in this example, is **rhubarb**. Once the same value is entered into the configuration for e*Way it should be able to connect and authenticate itself. Note that the default is **NOT SET**—unless this parameter is changed from the default value, the SEWS component **will not start**. The value you enter for the **Secret** parameter must match the value you enter later for the **Secret** parameter in the e*Way configuration. See **SWIFT Setup** on page 90.

5.3 Setting Up a Test Environment (Optional)

If you do not have an existing SWIFT environment suitable for testing, you can set up a simple loopback configuration for testing purposes, following installation of the SEWS component. When running this test environment, any e*Gate-outbound events sent to the SEWS component will be routed back as e*Gate-inbound events, using the **SEWS_from_Egate** and **SEWS_to_Egate** routing points.

To create this test environment, you must set up the following items to match your system and your schema.

1. Queues for the routing points.
2. The routing points themselves.
3. Logical terminals.
4. TCP/IP parameters.
5. Security options.

Remember to set your SWIFTAlliance servers to **housekeeping** mode. All procedures except those related to security options require only Operator privileges.

Note: *When you create a new schema, both the LSO and RSO must log in and **approve** it before you can **activate** it.*

5.3.1 Queues

First, you need to set up the queue for the routing points:

1. Launch the Systems Management window, invoke the **View** drop-down menu, and select **Queue**.
2. Open the routing point **SEWS_from_egate**, and select the Routing tab.
3. Under *Valid routing targets*, move **SEWS_to_egate** from **Available** to **Selected**.
4. **Modify** and **Save** this configuration.

5.3.2 Routing Points

Second, you need to configure the routing points to loop the messages from *inbound* back to *outbound*:

1. Launch the Routing window.
2. View the schemas to make sure your test schema is separate from any working schema. If not, create a dummy schema, then approve and activate it (see preceding *Note*).
3. With the test schema active, open the routing point **SEWS_from_egate** and create a new routing rule, setting the following parameters to the indicated values.

Parameter	Value
Condition On	Function Result and Message
Function Result	success
Message	Creating_mpfn = ‘SEWS_mpf’
Action On	source
Action	Route To (on the source sub-panel)
Route To	SEWS_to_egate
Append Intervention	No Intervention
Unit	Keep Current

4 Click **Validate**, then **Save** the configuration.

5 View the schema, and set it to **Approved** and **Active**.

5.3.3 Logical Terminals

Third, you need to set up the required logical terminals:

1 Launch the SWIFT Support window, invoke the **View** drop-down menu, and select **Mstv Id**.

2 Install 0105 and 0205.

3 Invoke the **View** drop-down menu again, and select **Logical Terminal**.

4 Create two new logical terminals having the following characteristics.

Destination	Terminal Code	Mstv_Id
SOTCBE2A	A	0105
STCOGB2L	A	0105

5 Launch the Security Definition window, invoke the **View** drop-down menu, and select **Component**.

6 Open the SEWS component, select the ADK tab, and move **none** from **Available** to **Selected**.

5.3.4 TCP/IP Parameters

Next, you need to set the TCP/IP parameters as described in **TCP/IP Parameters** on page 53.

5.3.5 Security Options

Finally, you need to set up the security options as described in **Security Options** on page 55.

5.4 Starting SEWS

From the **File** menu in the **System Management** window, select **Start Component** and then **SEWS**.

Chapter 6

Operational Overview

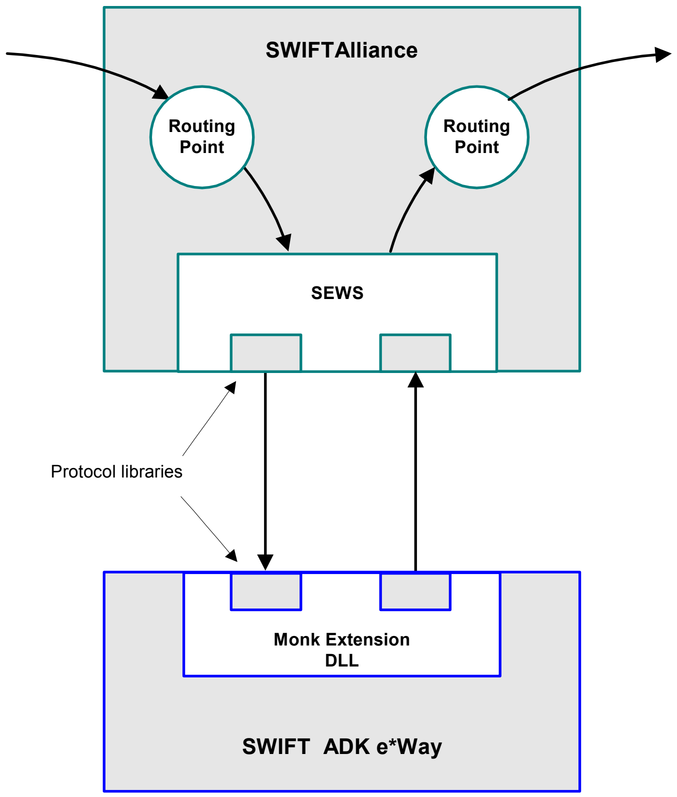
This chapter describes the basic operation of the SWIFT ADK e*Way and the way in which it interacts with SEWS.

6.1 Interacting with SWIFT

6.1.1 SWIFTAlliance, SEWS, and ADK

The SWIFT ADK e*Way uses the SWIFT client/server architecture by integrating the Alliance Developer Toolkit (ADK) into the e*Way environment. The ADK is a library of APIs that can call services provided by SWIFTAlliance servers.

Figure 31 SEWS and the SWIFT ADK e*Way



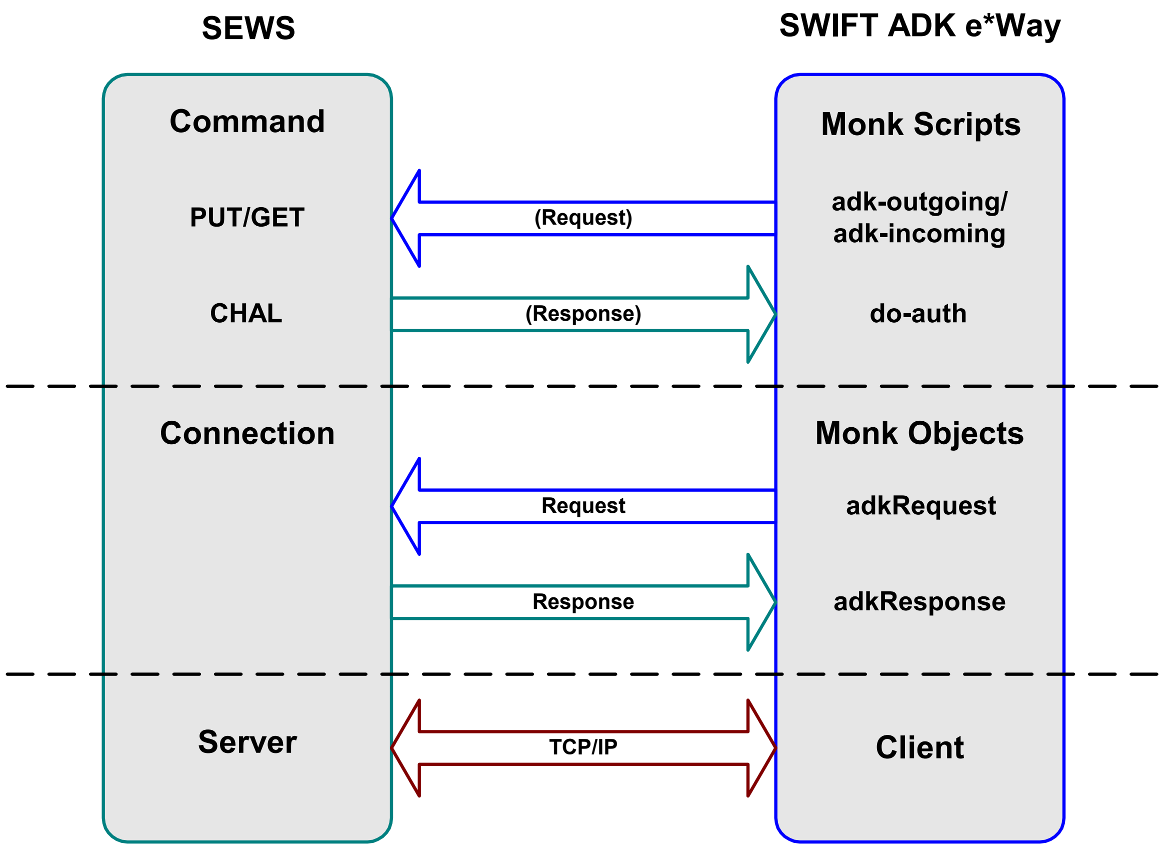
Within SWIFTAlliance, the SEWS component is linked to two routing points; one (**SEWS_to_egate**) receives Events from the SWIFT network and queues them for input to the SWIFT ADK e*Way; the other (**SEWS_from_egate**) is used for outgoing Events. These routing points are analogous to e*Gate Intelligent Queues (IQs).

SWIFT Events are identified by a unique identifier, composed of the **s_umid** and an instance number. The instance number distinguishes between separate Events having the same **s_umid** and content.

6.1.2 Communications Layers

Communication between SEWS and the SWIFT ADK e*Way can be viewed as having three layers, as depicted in Figure 32.

Figure 32 Communication Layers



Starting at the bottom, the lowest layer represents the TCP/IP network, and across it a TCP socket connection. At one end is the SEWS component acting as a server, at the other end the SWIFT ADK e*Way behaving as a client.

The middle layer represents the communication protocol used between the SEWS and the e*Way, which is a request/response protocol. The e*Way sends requests and receives responses from the SEWS via functionality provided by the **adkRequest** and **adkResponse** Monk objects. These objects can be thought of as containers for messages—they are constructed and manipulated in Monk, and unless they are populated by Monk code they are empty (and therefore meaningless).

The highest layer represents the protocol implemented on top of those **adkRequest** and **adkResponse** objects. These are Monk scripts used by the e*Way to create, populate and send requests and receive responses. The requests understood by the SEWS component have names, or request-types, like **PUT** (to upload a new message into SWIFT) and **GET** (to retrieve a message from SWIFT).

6.1.3 Event Flow

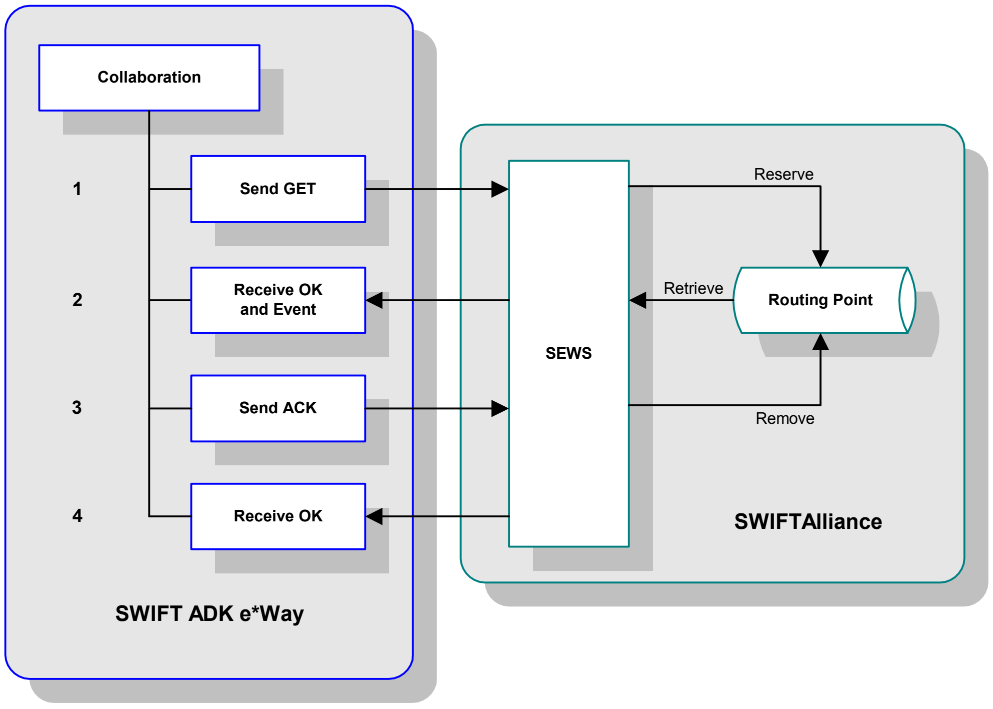
All communication is initiated by the SWIFT ADK e*Way; SWIFTAlliance does not send Events unless prompted to do so.

Inbound e*Way

Messages received from the SWIFT Network are stored in a routing point. In order to retrieve an Event from the routing point, the ADK component must first reserve it and then retrieve it. The Event remains in the routing point until the e*Way sends an acknowledgment to say that it has safely placed the Event in the appropriate IQ.

As an example, the following diagram shows the basic process flow involved in retrieving one Event from the e*Gate-inbound routing point. Note that **GET** and **ACK** are described in **"Monk ADK Functions" on page 114**.

Figure 33 SWIFT-to-e*Gate Basic Process Flow



1 The SWIFT ADK e*Way sends a **GET** message asking for the next available Event.

2 SEWS reserves the Event and retrieves it from the routing point.

3 The e*Way receives an **OK** reply to the **GET** with the Event in the content of the reply. It stores the Event in the appropriate IQ.

4 The e*Way **ACK**s SEWS to say that it has received and stored the Event.

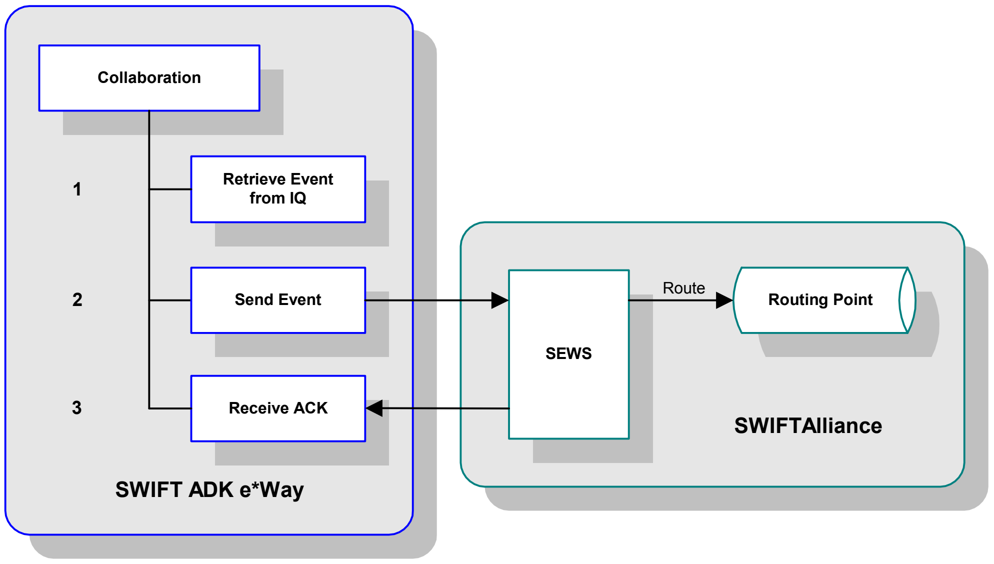
5 The Event is removed from the routing point.

6 The e*Way receives an **OK** reply to the **ACK**.

Outbound e*Way

An outgoing Event experiences a similar, but simpler, process:

Figure 34 e*Gate-to-SWIFT Basic Process Flow



1 The SWIFT ADK e*Way retrieves an Event from the IQ.

2 The e*Way sends the Event to SEWS, which routes it to the outgoing routing point.

3 SEWS sends an **ACK** to the e*Way, indicating success.

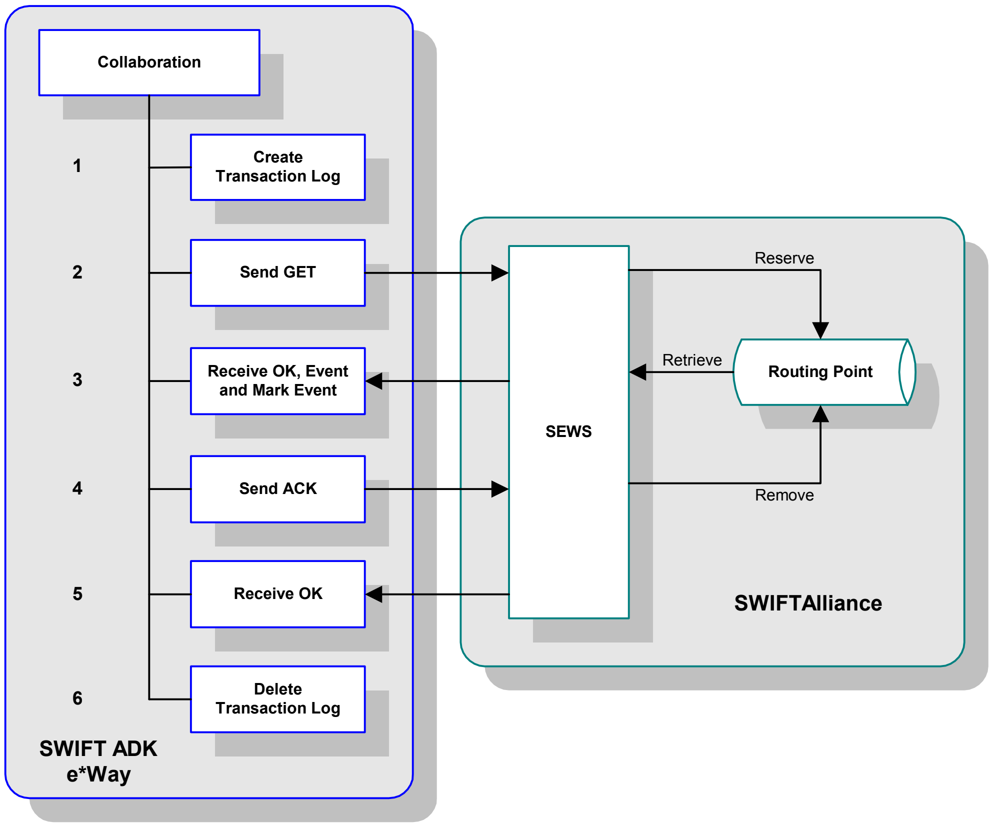
6.1.4 Data Integrity Features

When the e*Way starts up, it checks for indications of incomplete processing during a previous interchange.

Inbound e*Way

The process flow during initialization of an inbound e*Way is shown in Figure 35.

Figure 35 SWIFT-to-e*Gate Initial Process Flow



The SWIFT ADK e*Way creates a transaction log file, which is deleted when the e*Way shuts down. If a transaction log file already exists, the e*Way marks the next Event received from SEWS as a possible duplicate Event. The location of the log file is:

```
eGate/client/logs/ewIn.log
```

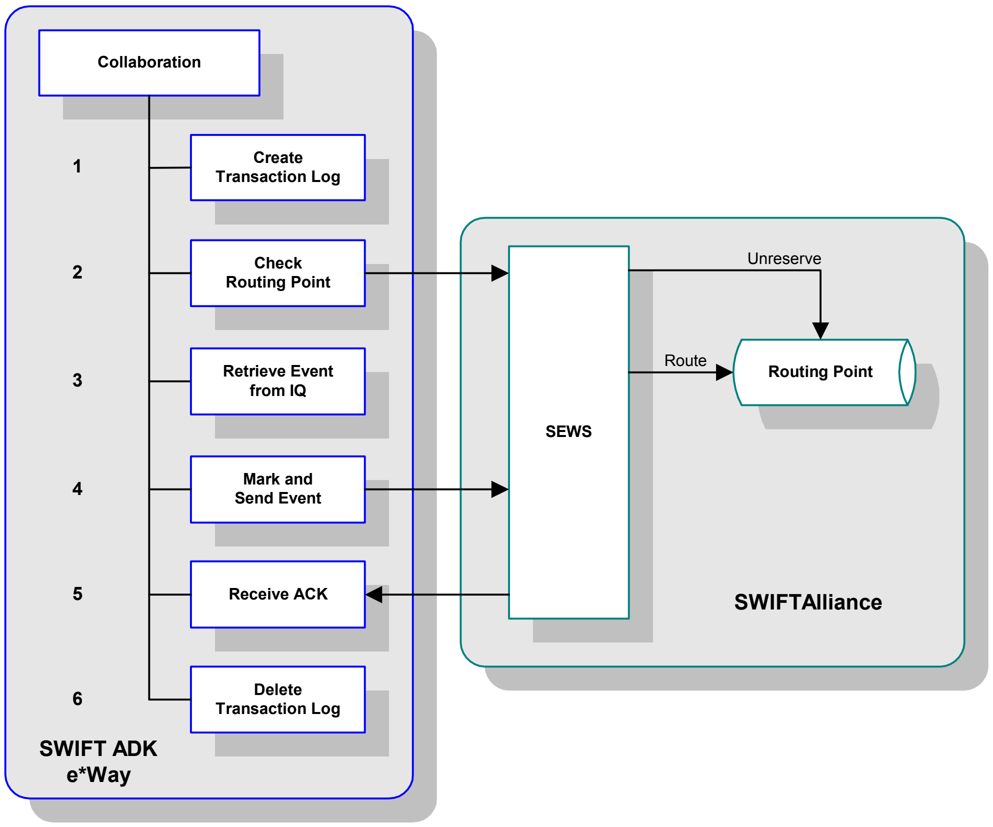
The e*Way also checks for reserved messages on the routing point. If the next Event on the routing point is reserved, the e*Way unreserves it and marks the next Event as a possible duplicate.

The remainder of the processing is handled normally (see Figure 33), as are all subsequent Events during the session.

Outbound e*Way

The procedure followed during initialization of an outbound e*Way is similar (see Figure 36).

Figure 36 e*Gate-to-SWIFT Initial Process Flow



The SWIFT ADK e*Way creates a transaction log file, which is deleted when the e*Way shuts down. If a transaction log file already exists, the e*Way marks the next Event received from the IQ as a possible duplicate Event. The location of the log file is:

```
eGate/client/logs/ewOut.log
```

The e*Way also checks for reserved Events on the routing point. If the next Event on the routing point is reserved, the e*Way unreserves it and routes it on to SWIFTAlliance. The e*Way then marks the next Event received from the IQ as a possible duplicate Event and sends the Event to SEWS.

The remainder of the processing is handled normally (see Figure 34), as are all subsequent Events during the session.

6.1.5 Diagnostics and Recovery

SWIFTAlliance Failure

In case of a SWIFTAlliance failure, the **ACK/NAK** protocol recovers automatically, and no messages are lost.

SEWS Failure

In case of a SEWS failure, the e*Way automatically detects the lost connection, and SWIFTAlliance also provides an alert. If desired, SWIFTAlliance can be configured to automatically restart SEWS; otherwise, you need to restart SEWS manually by means of the system management facility.

The status of the SEWS component can be checked via the e*Gate Monitor (connection up or down), and using the standard SWIFTAlliance facilities (system monitor and the system management GUI).

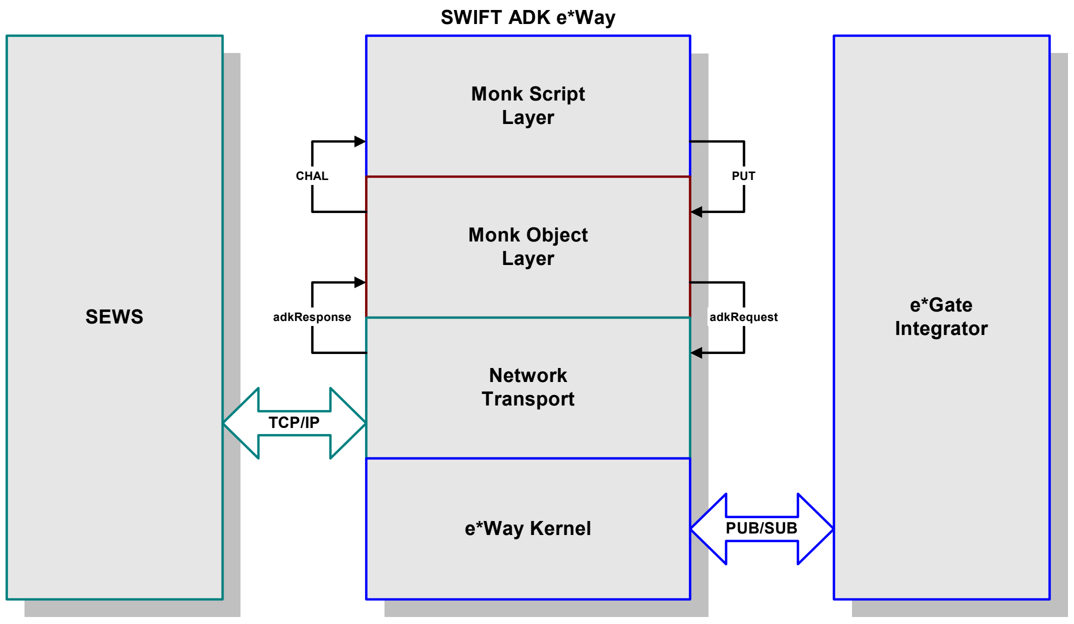
e*Way Failure

If the SWIFT ADK e*Way fails, the TCP/IP connection is broken and SEWS enters a *waiting for connection* state. The e*Way is then restarted manually or automatically, depending upon its configuration. Resolution of unsent or unacknowledged messages occurs automatically (see previous section).

6.2 SWIFT ADK 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 SWIFT ADK e*Way Architecture

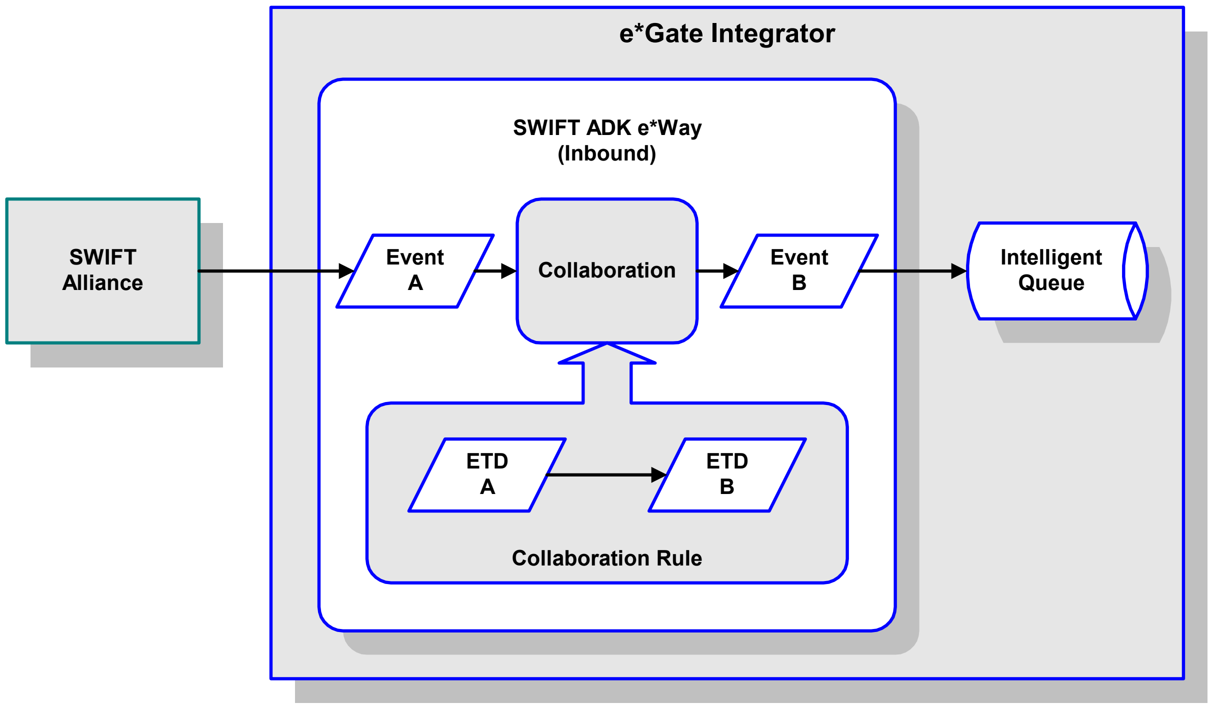


The upper layers of the e*Way use Monk functions to perform Business Process modeling and ETD mapping, package data as e*Gate *Events*, send those Events to Collaborations, and manage interaction with the external system. These layers are built upon an e*Way Kernel layer that manages the basic operations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

The communication layers of the e*Way are single-threaded. Functions run serially, and only one function can be executed at a time. Processing layers are multi-threaded, with one executable thread for each Collaboration. Each thread maintains its own Monk environment; therefore, information such as variables, functions, path information, and so on cannot be shared between threads.

Collaborations execute the business logic that enable the e*Way to do its intended work. In turn, each Collaboration executes a Collaboration Rule, containing the actual instructions to execute the business logic. Each Collaboration that publishes its processed Events internally (within e*Gate Integrator) requires one or more IQs to receive the Events, as shown in Figure 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 these basic e*Way operations are discussed in **Chapter 7**. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as *Microsoft Word*, *Notepad*, or UNIX *vi*). The available set of e*Way API functions is described in **Chapter 8**. 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.

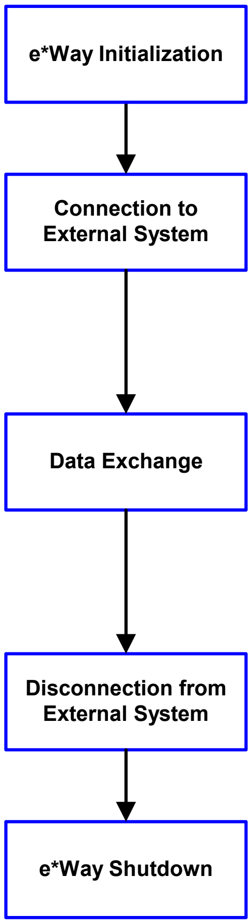
For more information on defining Collaborations, defining IQs, assigning Collaborations to e*Ways, or configuring Collaborations to publish Events, see the *e*Gate Integrator User's Guide*.

6.3 Basic e*Way Processes

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

The most basic processes carried out by an e*Way are listed in Figure 39. In e*Ways based on the Generic Monk e*Way Kernel (using **stcewgenericmonk.exe**), these processes are controlled by the listed Monk functions. Configuration of these functions is described in the referenced sections of this User's Guide.

Figure 39 Basic e*Way Processes


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

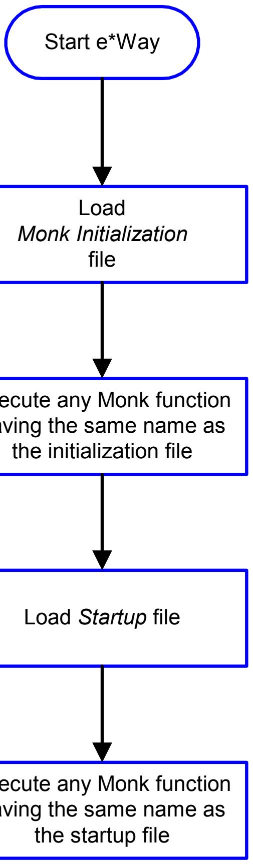
Process	Monk Configuration Sections
e*Way Initialization	Startup Function on page 83 (also see Monk Environment Initialization File on page 83)
Connection to External System	External Connection Establishment Function on page 86 External Connection Verification Function on page 86
Data Exchange	**Event-driven Data Exchange** Process Outgoing Message Function on page 84 **Schedule-driven Data Exchange** Exchange Data with External Function on page 85 Positive Acknowledgment Function on page 87 Negative Acknowledgment Function on page 88
Disconnection from External System	External Connection Shutdown Function on page 87
e*Way Shutdown	Shutdown Command Notification Function on page 89

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 7, while the functions themselves are described in Chapter 8.

Initialization Process

Figure 40 illustrates the e*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

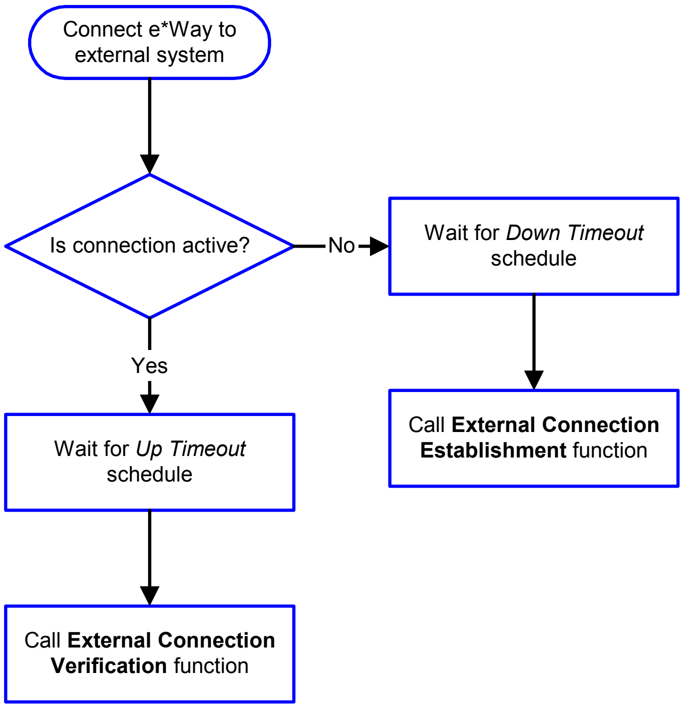
Figure 40 Initialization Process


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

Figure 41 Connection Process


Connect e*Way to external system
Is connection active?
No
Wait for *Down Timeout* schedule
Call **External Connection Establishment** function
Yes
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 43 on page 73** *and* **Figure 42 on page 72** *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 127 and* **send-external-down** *on page 127 for more information.*

Data Exchange Process

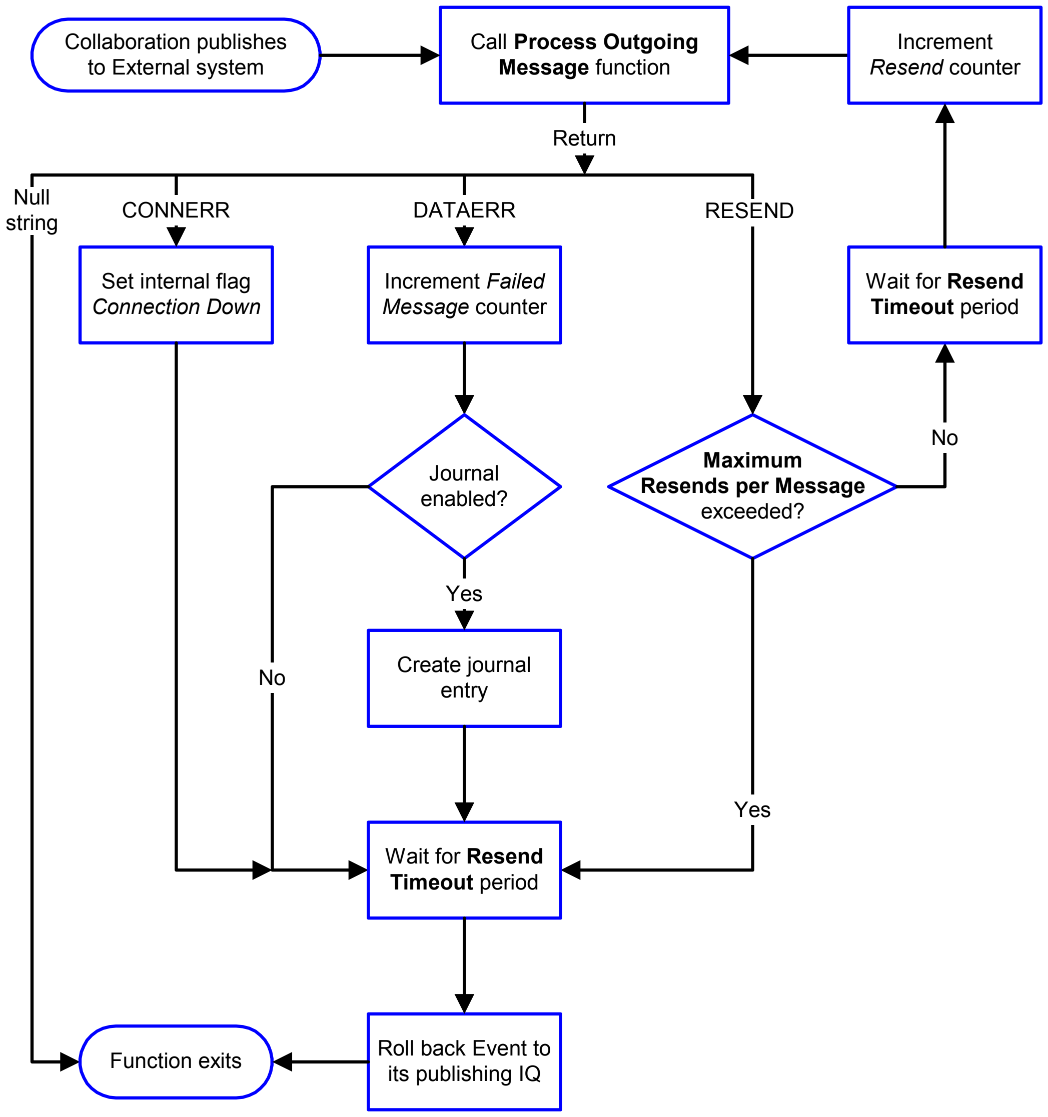
Event-driven

Figure 42 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.

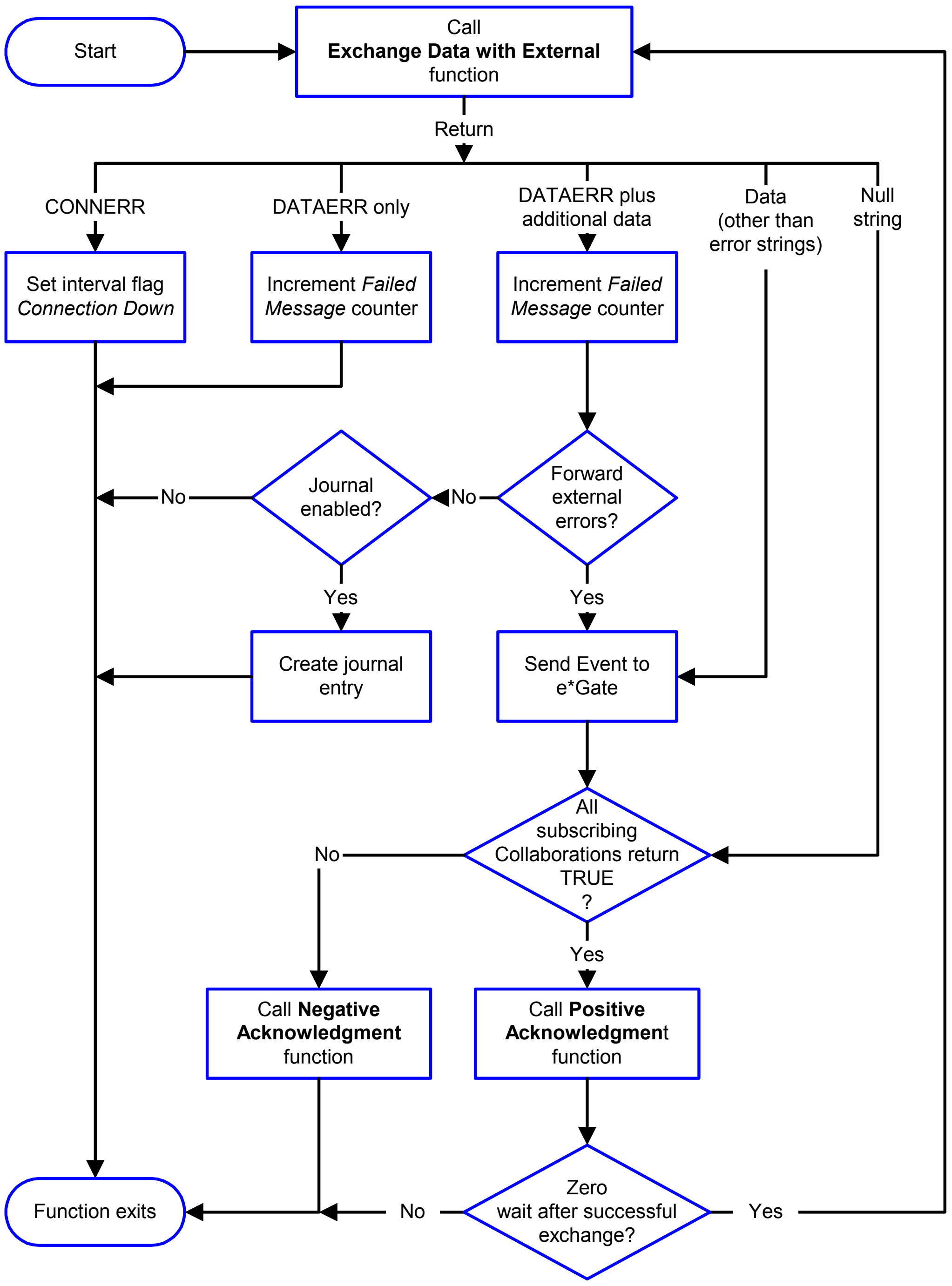
Figure 42 Event-Driven Data Exchange Process


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
Journal enabled?
Maximum Resends per Message exceeded?
No
Yes
Create journal entry
No
Yes
Wait for Resend Timeout period
Function exits
Roll back Event to its publishing IQ

Schedule-driven

Figure 43 illustrates how the e*Way's schedule-driven data exchange process works for incoming data, using the **Exchange Data with External Function**, **Process Outgoing Message Function**, and **Negative Acknowledgment Function**.

Figure 43 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

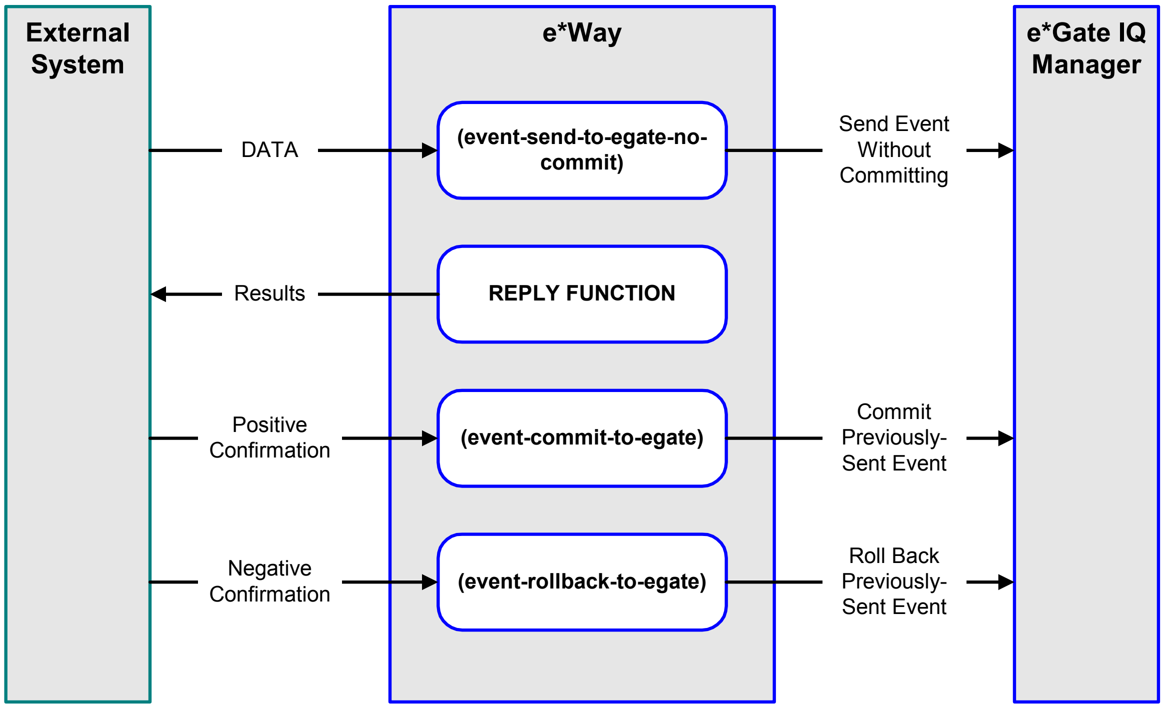
*Send Events to e*Gate* can be implemented using any of the following Monk functions:

- **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 44):

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

Figure 44 Send Event to e*Gate with Confirmation

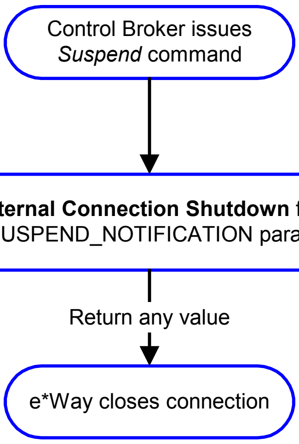

External System
e*Way
e*Gate IQ Manager
DATA
(event-send-to-egate-no-commit)
Send Event Without Committing
Results
REPLY FUNCTION
Positive Confirmation
(event-commit-to-egate)
Commit Previously-Sent Event
Negative Confirmation
(event-rollback-to-egate)
Roll Back Previously-Sent Event

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

Disconnect from External Process

Figure 45 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

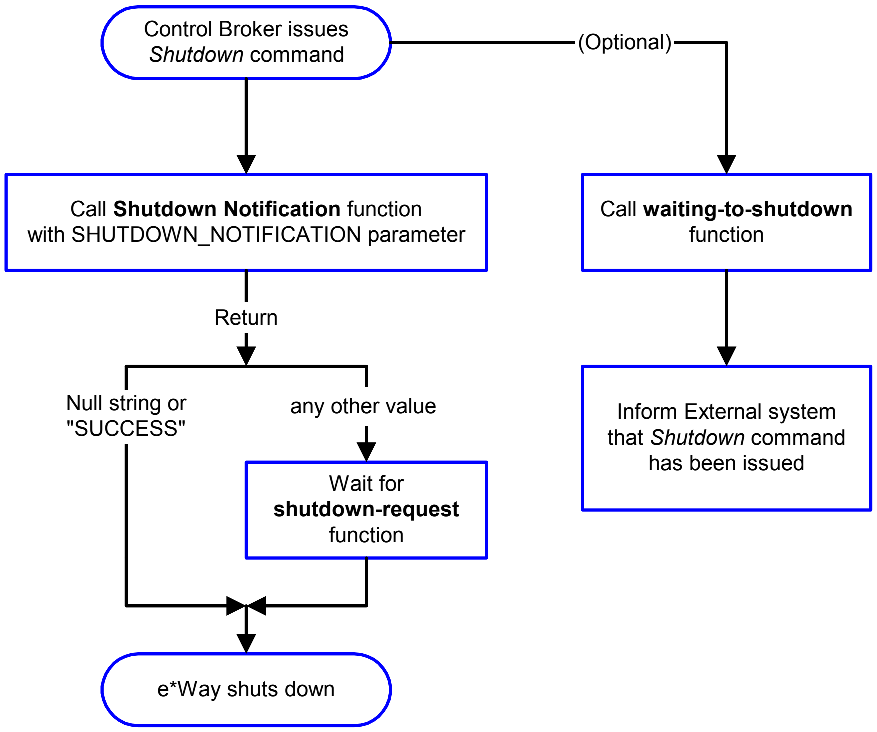
Figure 45 Disconnect Process


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

Shutdown Process

Figure 46 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

Figure 46 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 7

Configuration Parameters

This chapter describes the configuration parameters for the SWIFT ADK e*Way.

7.1 Overview

The e*Way's configuration parameters are set using the e*Way Editor; see **Configuring the e*Way** on page 42 for procedural information. The default configuration is provided in **SwiftADK.def**. The SWIFT ADK e*Way's configuration parameters are organized into the following sections:

General Settings on page 77

Communication Setup on page 79

Monk Configuration on page 82

SWIFT Setup on page 90

7.2 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid filename, optionally including an absolute path (e.g., **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 written to the journal file 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 (Events) that the e*Way allows. When the specified number of failed messages is reached, the e*Way shuts down and exits.

Required Values

An integer between **1** and **1,024**. The default is **3**.

Forward External Errors

Description

Specifies whether or not error messages that begin with the string **"DATAERR"** that are received from the external system is queued to the e*Way's configured queue. See **Exchange Data with External Function** on page 85 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 72 for more information about how the e*Way uses this function.

7.3 Communication Setup

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

Note: *The schedule you set using the e*Way's properties in the 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 is exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

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

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

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

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an ACK or a NAK to the external system (using the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively) and whether or not the connection to the external system is active. If no ACK or NAK is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External Function**. Thereafter, the **Exchange Data with External Function** is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

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

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 parameter is ignored and the e*Way immediately invokes the **Exchange Data with External Function**.

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

See also

Down Timeout on page 80

Stop Exchange Data Schedule on page 79

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 (Event) to the external system, after receiving an error message from the external system.

Required Values

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

Zero Wait Between Successful Exchanges

Description

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

Required Values

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

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

7.4 Monk Configuration

The parameters in this section help you set up the information required by the e*Way to utilize Monk for communication with the external system.

Specifying Function or File Names

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

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

- .monk
- .tsc
- .dsc

Specifying Multiple Directories

To specify multiple directories, manually enter the directory names rather than selecting them with the **File Selection** button. Directory names must be separated with semicolons, and you can mix absolute paths with relative e*Gate paths. For example:

```
monk_scripts\my_dir;c:\my_directory
```

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

Load Path

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

Additional Path

Description

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

Required Values

A pathname, or a series of paths separated by semicolons. There is no default value for this parameter.

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

Additional information

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

Auxiliary Library Directories

Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories is automatically loaded into the e*Way's Monk environment. The default value is **monk_library/swiftadk**.

Required Values

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

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

Monk Environment Initialization File

Description

Specifies a file that contains environment initialization functions, which is loaded after the **Auxiliary Library Directories** are loaded.

Required Values

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

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 **adk-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. The default value is **adk-outgoing**.

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 **Down Timeout** 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. The default value is **adk-incoming**.

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 **adk-connect**.

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.

Required Values

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

Returns

- **"SUCCESS"** or **"UP"** indicates that the connection was established successfully
- Any other string (including the null string) indicates that the attempt to establish the connection failed

Additional Information

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

External Connection Shutdown Function

Description

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **adk-disconnect**.

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 value is **adk-ack**.

Required 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 value is **adk-nak**.

Required 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. The default value is **adk-shutdown**.

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.

7.5 SWIFT Setup

The parameters in this section specify the information required by the e*Way to connect to SEWS on the SWIFTAlliance system.

Hostname

Description

The Name or IP Address of the host on which the SEWS component is running.

Required Values

A string.

Note: *This value must match the* **IP Address** *value entered for SWIFTAlliance/SEWS.*

Port Number

Description

The TCP Port Number on which the SEWS component is listening for new connections

Required Values

Select a Port Number **1** through **65535**. The default is **14000**.

Note: *This value must match the* **TCP Port Number** *value entered for SWIFTAlliance/SEWS.*

Key

Description

The value of this parameter is used to encrypt the **Secret** parameter (see following entry) at configuration time (versus run time). It is used as soon as the user depresses the ENTER key while in the **Secret** field; therefore, the latter must be re-encrypted if the value of the **Key** parameter ever changes.

Required Values

A string.

Note: *If the value of the* **Key** *parameter is changed, the value of the* **Secret** *parameter must be re-entered, even if the value of the* **Secret** *parameter is unchanged.*

Secret

Description

This parameter is used for authentication of connecting clients.

Required Values

A string. The value must match the **Secret** value in the SWIFT Alliance Security Definition configuration for the SEWS component.

Note: *If the value of the* **Key** *parameter is changed, the value of the* **Secret** *parameter must be re-entered, even if the value of the* **Secret** *parameter is unchanged.*

Message Validation Level

Description

This parameter allows selection of the following validation levels:

- **Maximum**

  Currently the same as **Intermediate**, this option will be enhanced to provide a higher level of validation in a future release.

- **Intermediate**

  Provides syntactic validation of the message text at field level (for example: presence of mandatory fields, keyword validation, limits, and ranges of values).

- **Minimum**

  Provides validation only for the control block structure of the message (if any).

- **None**

  Provides no validation at all. This option is valid only if Express Traffic is licensed for the e*Way's target SWIFTAlliance system.

Required Values

One of the options described above.

Note: *If the external SWIFTAlliance system does* ***not*** *have an Express Traffic license, and the e*Way is operated with a validation level of* **None***, the e*Way will throw an exception each time it attempts to send a message to SWIFTAlliance.*

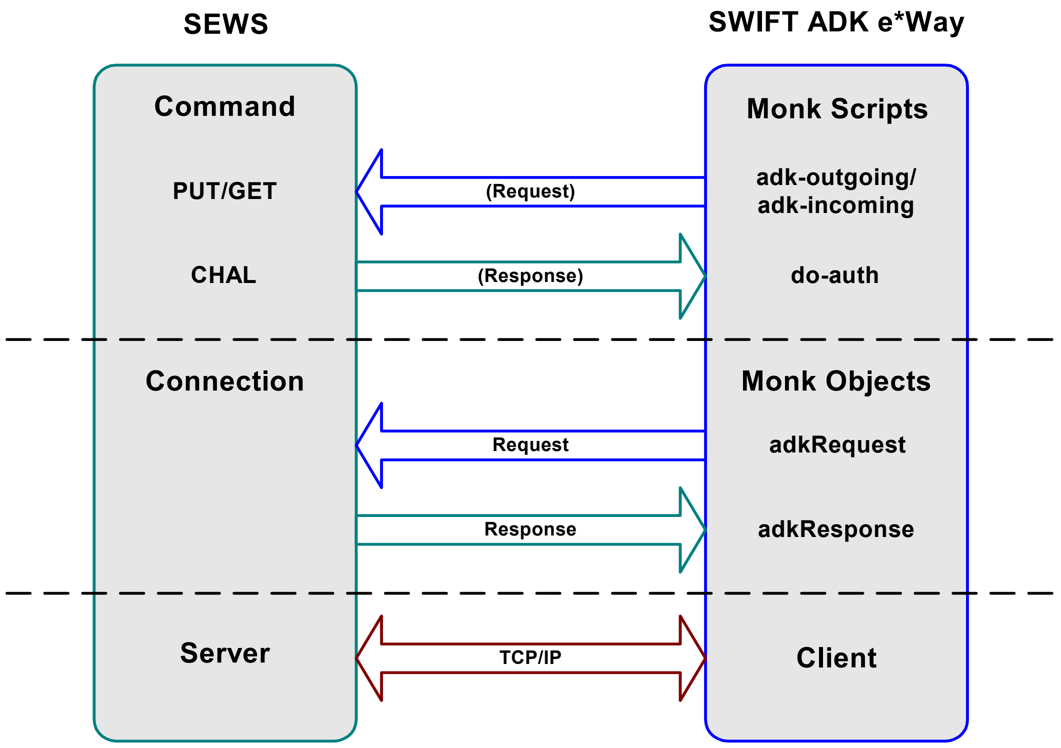
Chapter 8

API Functions

8.1 Overview

Most of the SWIFT ADK e*Way API functions are used in the various communication layers that link the e*Way with ‘SEWS (see Figure 47).

Figure 47 Communication Layers



These functions are categorized as follows (refer to Figure 47):

- **SEWS Component Protocol** on page 93, corresponding to the SEWS Command set.
- **Monk Extension Methods** on page 102, corresponding to the Monk objects.
- **Monk ADK Functions** on page 114, corresponding to the Monk scripts.
- **Generic e*Way Functions** on page 123, contained in the Generic e*Way kernel.

8.2 SEWS Component Protocol

This section describes the SEWS component protocol contained within the SWIFT Monk extension dynamic load library, **stc_swiftadk.dll**.

ACK

Description

Acknowledges that the last message received from a GET request has been successfully committed to disk.

ACK messages do not contain a Content.

Request Properties

Name	Usage	Description
s_umid	Mandatory	The unique identifier of the message being acknowledged.
instance	Optional	The instance identifier of the message being acknowledged. The default is the instance number of the last message to be received.
rp_name	Optional	The routing point to acknowledge on: ▪ SEWS_to_egate ▪ SEWS_from_egate The default depends on the direction at authentication: SEWS_to_egate if the direction is "to_egate", and SEWS_from_egate otherwise.
force	Optional (value = **y**)	Use this parameter, a value of **y**, to acknowledge a message that was *not* the last one received with a GET request.

Returns

Success:

Value	Description
s_umid	The unique identifier of the message that has been acknowledged.
instance	The instance identifier of the message that has been acknowledged.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **502**, **504**, **505**, **507**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

AUTH

Description

Allows the Monk client to authenticate itself to SEWS (SEWS does not process any requests from an unauthorized client).

The authentication procedure avoids sending readable passwords over the network via a *challenge-authentication* mechanism. Both sides (client and SEWS) hold a password, (referred to as the **secret**). When the client connects to SEWS, it receives a **CHAL** response with a **challenge** argument, containing a random number.To correctly authenticate itself, the client needs to make an **AUTH** request containing the MD5 hash of the concatenation of the **secret** and the **challenge**.

Functions to perform the MD5 hash calculation are available in **stc_monkadk.dll**. See **adkMD5 class** on page 112.

AUTH messages do not contain a Content.

Request Properties

Name	Usage	Description
direction	Optional	Specifies the direction in which data flows for this connection: ▪ to_egate ▪ from_egate (default) Single clients can be bidirectional, by implementing 2 connections.
response	Mandatory	The MD5 hash of the concatenation of the secret and the challenge.

Returns

Success:

String containing the Content “**Authorization OK**”.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **519** and **581**. Content contains a description of the error. See **Error Codes** on page 32.

CHAL

Description

Response sent to the client when the connection is first established. It is always positive.

CHAL messages do not contain a Content.

Request Properties

Name	Usage	Description
s_umid	Mandatory	A unique identifier (a random number) contained in the challenge value, used in the authentication procedure. See **AUTH** on page 94.

COUNT

Description

Counts the number of messages on the SWIFTAlliance routing point. All messages, not only those available for processing can be included in the count.

COUNT messages do not contain a Content.

Request Properties

Name	Usage	Description
state	Optional	Controls which type of messages are to be counted: ▪ reserved (default) ▪ unreserved ▪ all

Name	Usage	Description
rp_name	Optional	This parameter defines the routing point to count. By default, the two routing points associated with SEWS are “SEWS_to_egate” and “SEWS_from_egate”, but these may be different if a non-standard installation has been performed. The two routing points are automatically passed to the SEWS executable on startup by the -i and -o flags. If this option is not supplied, the default of “SEWS_to_egate” (or the value of the -i startup argument) is used.

Returns

Success:

A vector containing the number of matching messages found.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **501**, **511**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

GET

Description

Reserves and retrieves a message from SWIFTAlliance.

GET messages do not contain a Content.

Note: *Before a message is retrieved, all previously retrieved messages must be acknowledged with an* **ACK** *request. Additionally, only un-reserved messages can be retrieved; therefore, it is advisable to use a* **LIST** *request before a* **GET***. See* **setHeader** *on page 108.*

Request Properties

A message is identified in SWIFTAlliance with a **s_umid** and an instance number. The **s_umid** identifies an unique message, where there are one or more instances of that message inside SWIFTAlliance, all with the same content, but at different locations within SWIFTAlliance.

Name	Usage	Description
s_umid	Optional	The unique identifier of the message to be retrieved. If not specified, GET defaults to the next unreserved message.
instance	Optional	The instance number of the given s_umid. If not specified, the instance defaults to zero (0).

Returns

Success:

Value	Description
s_umid	The unique identifier of the message just received.
type	The message type of the retrieved message: Swift or Telex.

(Content contains the text of the retrieved message.)

Failure:

Value	Description
s_umid	The unique identifier of the message requested.
instance	The instance number of the request s_umid.
code	The following codes can be returned: **201**, **507**, **509**, **510**, and **516**. Content contains a description of the error. See **Error Codes** on page 32.

JOURNAL

Description

Enables a message to be passed to SWIFTAlliance, to be logged as human-readable log events.

The Content is the actual message to be logged in SWIFTAlliance.

Request Properties

Name	Usage	Description
level	Mandatory	The level of event log. The allowed values for this property are: **info**, **warning**, and **error**.

LIST

Description

Returns a list of **s_umids** on a routing point. The message listing is given as a list of SWIFT (**s_umid** instance number) pairs, joined with a space and with one pair per line.

LIST messages do not contain a Content.

Request Properties

Name	Usage	Description
state	Optional	Controls which type of messages are to be listed: ▪ reserved (default) ▪ unreserved ▪ all
rp_name	Optional	This parameter defines the routing point to listed. By default, the two routing points associated with SEWS are “SEWS_to_egate” and “SEWS_from_egate”, but may be different if a non-standard installation has been performed. The two routing points are automatically passed to the SEWS executable on startup by the -i and -o flags. If this option is not supplied, the default of “SEWS_to_egate” (or the value of the -i startup argument) is used.

Returns

Success:

A vector containing a list of **s_umid** instance pairs, one pair per line.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **501**, **511**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

NAK

Description

Sends a negative acknowledgment to SWIFTAlliance, indicating that the last message received from a **GET** request could not be processed successfully.

NAK messages do not contain a Content.

Request Properties

Name	Usage	Description
s_umid	Mandatory	The unique identifier of the message being negatively acknowledged.
instance	Optional	The instance identifier of the message being negatively acknowledged. The default is the instance number of the last message to be received.
rp_name	Optional	The routing point to acknowledge on: ▪ SEWS_to_egate (default) ▪ SEWS_from_egate
force	Optional (**y**)	Use this parameter, a value of **y**, to acknowledge a message that was *not* the last one received with a GET request.

Returns

Success:

Value	Description
s_umid	The unique identifier of the message that was negatively acknowledged.
instance	The s_umid instance of the message that was negatively acknowledged.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **502**, **504**, **505**, **507**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

PUT

Description

Inserts messages into SWIFTAlliance via SEWS.

Request Properties

Name	Usage	Description
type	Mandatory	The message type for the new message: Swift or Telex. The content contains the text of the message to be sent.

Returns

Success:

Name	Description
s_umid	The unique identifier of the inserted message.

Failure:

Name	Description
offset	Contains the string offset where the validation failed.
reason	Contains a string identifying the error.
code	The following codes can be returned: **506**, **513**, **514**, **515**, **550**, and **551**. Content contains a description of the error. See **Error Codes** on page 32.

RECOVER

Description

Recovers reserved messages on the SEWS routing point.

- For inbound messages, it unreserves any previously-reserved message on the **SEWS_to_egate** routing point and marks the next incoming message as a potential duplicate.
- For outbound messages, it unreserves any previously-reserved message on the **SEWS_from_egate** routing point, routes it on to SWIFTAlliance, and marks the next outgoing message as a potential duplicate.

RECOVER messages do not contain a Content.

Request Properties

Name	Usage	Description
s_umid	Mandatory	The unique identifier of the message to be recovered.
instance	Optional	The instance identifier of the message to be recovered. The default is **0**.
rp_name	Optional	The routing point to recover on: ▪ SEWS_to_egate ▪ SEWS_from_egate The default depends upon the direction at the time of authentication: ▪ SEWS_to_egate, if direction is “to_egate” ▪ SEWS_from_egate, if otherwise

Returns

Success:

Name	Description
s_umid	The unique identifier of the recovered message.
instance	The instance identifier of the recovered message.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **502**, **506**, **508**, **509**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

8.3 Monk Extension Methods

This section describes the object methods contained in the Monk extension dynamic load library, **stc_monkext.dll**. The Monk Extension is a "wrapper" for the Protocol library that implements the communication between the Monk extension library and SEWS.

The methods are organized according to the following object classes:

8.3.1 adkConnection class

This class models the connection to SEWS within the SWIFTAlliance server. It provides a way to connect to the SWIFTAlliance server and pass the connection object as an argument to the **adkRequest** and **adkResponse** **make** and **take** functions.

The methods in this class are:

Example of adkConnection Methods

```
(display "creating adkConnection object ..." )
(define cnx (load-interface "stc_swiftadk.dll" "adkConnection_init" )
                     )
(display " done.\n" )

(define host "localhost")
(define port 16600)
(display "connecting to " ) (display host) (display ":" ) (display
                     port) (display "... " )
(invoke cnx "connect" host port)
(display "Connected\n" )
```

constructor

As with all object classes accessed via the Monk load-interface function, the **constructor** takes a fixed set of arguments.

Signature

```
(define your-object (load-interface "stc_swiftadk.dll"
                     "adkConnection_init" ) )
```

connect

Description

Joins the e*Way to SEWS on the host named *host* using the port number *port*. The host and post number can be set in the e*Way Editor.

Signature

```
(object “connect” _host, port )
```

Parameters

Name	Type	Description
_host	String	The name of the host to which to connect.
port	Integer	The port number to use for the connection.

Returns

None.

Throws

If the connection is not successful, the method throws a Monk exception.

disconnect

Description

Severs the connection from SEWS.

Signature

```
(object “disconnect” )
```

Parameters

None.

Returns

None.

Throws

If the disconnection is not successful, the method throws a Monk exception.

8.3.2 adkRequest class

The **adkRequest** object class allows the Monk developer to compose and send requests to SEWS within the SWIFTAlliance installation.

Each **adkRequest** object has several salient features:

- **Header**

  A single line of text *without* carriage-return or line-feed characters. The header is used for words such as **GET**, **PUT** or **EXECUTE**.

- **Dictionary**

  The **adkRequest** object contains a string-to-string association, also known as a dictionary. The left side of the association is the *key* part while the right side is the *value* part. Neither the *key* nor the *value* string can contain carriage-return or line-feed characters, and the *key* cannot contain the colon character (:). There is no practical limit on the length of the *key* or *value*.

  The dictionary is optional and is used to store optional arguments for the header.

Note: *The key* **content-length** *is reserved for use in transmission. We recommend that you avoid using it.*

- **Content**

  A free-form string that can contain any character, including null characters. It is restricted to a maximum length of 4 Gigabytes.

The methods in this class are:

constructor on page 105	**getValue** on page 107
make on page 105	**setValue** on page 107
take on page 105	**getHeader** on page 108
asString on page 106	**setHeader** on page 108
keysValue on page 106	**getContent** on page 109
existsValue on page 106	**setContent** on page 109

Example of adkRequest Methods

```
(display "creating adkRequest object ..." )
(define req (load-interface "stc_swiftadk.dll" "adkRequest_init" ) )
(display " done.\n" )

(req "setHeader" "random header data" )
(req "setValue" "foo" "12" )
(req "setContent" "some content data ...\n... split over two lines" )
(req "make" cnx )
```

constructor

As with all object classes accessed via the Monk load-interface function, the **constructor** takes a fixed set of arguments.

Signature

```
(define your-object (load-interface "stc_swiftadk.dll"
                     "adkRequest_init" ) )
```

make

Description

Sends the request to SEWS, using the **connection** argument.

Signature

```
(obj "make" connection )
```

Parameters

Name	Type	Description
connection	Monk object	Must be an **adkConnection** object. See **disconnect** on page 103.

Returns

None.

Throws

If the transmission is not successful, the method throws a Monk exception.

take

Description

Allows the Monk developer to receive an **adkRequest** object from the specified **adkConnection** connection.

Signature

```
(obj "take" connection )
```

Parameters

Name	Type	Description
connection	Monk object	Must be an **adkConnection** object. See **disconnect** on page 103.

Returns

None.

Throws

If an error occurs, the method throws a Monk exception.

Additional Information

This method is included for completeness only, and may not be included in later releases.

asString

Description

Expresses the **adkRequest** object as a string.

Signature

```
(obj "asString" )
```

Parameters

None.

Returns

A vector containing a string representation of the **adkRequest** object.

Throws

None.

keysValue

Description

Provides a listing of the *key* strings in the **dictionary** component.

Signature

```
(obj "keysValue" )
```

Parameters

None.

Returns

A vector containing a list of strings.

Throws

None.

existsValue

Description

Tests for the existence of a *value* associated with the specified *key* in the **adkRequest** dictionary.

Signature

```
(obj "existsValue" key )
```

Parameters

Name	Type	Description
key	String	Name of the dictionary entry to be tested.

Returns

A vector containing a Boolean true **(#t)** if the *key* exists in the dictionary, and has a *value* associated with it; otherwise, a Boolean false (#f)

Throws

None.

getValue

Description

Retrieves the *value* associated with the specified *key* in the **adkRequest** dictionary.

Signature

```
(obj "getValue" key )
```

Parameters

Name	Type	Description
key	String	Name of the dictionary entry to retrieve.

Returns

A vector containing the *value* associated with *key* from the dictionary; if no *value* exists for the given *key*, then a vector containing an empty string is returned.

Throws

None.

setValue

Description

Sets *key* to the specified *value* in the **adkRequest** dictionary, overwriting any existing *value*.

Signature

```
(objt "setValue" key, value )
```

Parameters

Name	Type	Description
key	String	Name of an entry in the dictionary.
value	String	Value to associate with the given key.

Returns

None.

Throws

None.

getHeader

Description

Retrieves the value of the **adkRequest** header component.

Signature

```
(obj "getHeader" )
```

Parameters

None.

Returns

A vector containing a string representation of the header value.

Throws

None.

setHeader

Description

Sets the value of the **adkRequest** header component.

Signature

```
(obj "setHeader" header )
```

Parameters

Name	Type	Description
header	String	The new header string.

Returns

None.

Throws

None.

getContent

Description

Retrieves the value of the **adkRequest** content.

Signature

```
(obj "getContent" )
```

Parameters

None.

Returns

A vector containing a string representation of the **adkRequest** content.

Throws

None.

setContent

Description

Sets the value of the *content* component of the **adkRequest** to *content*.

Signature

```
(obj "setContent" content )
```

Parameters

Name	Type	Description
content	String	The new content value.

Returns

None.

Throws

None.

8.3.3 adkResponse class

This object class models responses to **adkRequest** objects sent to SEWS. Conceptually, it is very similar to the **adkRequst** class, because the underlying C++ object derives from it. The key difference is a success/fail value, which can be manipulated via the **setOK** and **getOK** methods.

The methods in this class are:

Example of adkResponse Methods

```
(display "creating adkResponse object ..." )
(define resp (load-interface "stc_swiftadk.dll" "adkResponse_init" ))
(display " done.\n" )

(resp "take" cnx)
(display (resp "asString" ))(newline)
(display "content is \"") (resp "getContent") (display "\"\n" )
```

constructor

As with all object classes accessed via the Monk load-interface function, the **constructor** takes a fixed set of arguments.

Signature

```
(define your-object (load-interface "stc_swiftadk.dll"
                     "adkResponse_init" ) )
```

getOK

Description

Retrieves a representation of the success or failure value of the **adkResponse** object.

Signature

```
(obj “getOK”)
```

Parameters

None.

Returns

A vector containing a Boolean true (**#t**) for success, or a Boolean false (**#f**) for failure.

Throws

None.

Additional Information

The function **isOk** can be used synonymously.

setOK

Description

Sets the success or failure value of the **adkResponse** object.

Signature

```
(obj “setOK” ok-value )
```

Parameters

Name	Type	Description
ok-value	Boolean	Boolean true (#t) or false (#f).

Returns

None.

Throws

None.

8.3.4 adkMD5 class

These methods are used for calculating MD5 hash values.

The methods in this class are:

Example of Using adkMD5 Methods

```
(monk-flag-set 'all)
(define input "Finbar Saunders & His Double Entendres" )

(define obj (load-interface "stc_swiftadk.dll" "adkMD5_init" ) )
(newline)
(display input ) (display "\n... becomes ...\n" )
(display (vector-ref (obj "calculate" input ) 0))
(newline)
```

constructor

As with all object classes accessed via the Monk load-interface function, the **constructor** takes a fixed set of arguments.

Signature

```
(define your-object (load-interface "stc_swiftadk.dll"
                     "adkMD5_init" ) )
```

calculate

Description

Computes the hash value from the *input* string.

Signature

```
(obj “calculate” input )
```

Parameters

Name	Type	Description
input	String	Input string to be used to calculate the hash value.

Returns

A vector containing the hash value of the *input* string.

Throws

If an error occurs, a Monk exception is thrown.

last

Description

Retrieves the last MD5 hash value calculated.

Signature

```
(obj “last” )
```

Parameters

None.

Returns

A vector containing the last hash value calculated.

Throws

None.

usage

Description

Obtains the number of times this object has been used.

Signature

```
(obj “usage” )
```

Parameters

None.

Returns

A vector containing an integer representing the number of times the object has been used.

Throws

None.

8.4 Monk ADK Functions

This section describes the functions available in Monk scripts within the highest communications layer. They contain references to, and make use of, the SEWS component protocols residing within the SWIFT Monk extension DLL.

The commands are:

adk-ack

Description

Sends a positive acknowledgment to SEWS that the last message received from a **GET** request has been successfully committed to disk.

Signature

```
(adk-ack s_umid instance rp_name force)
```

Parameters

Name	Usage	Description
s_umid	Mandatory	The unique identifier of the message being acknowledged
instance	Optional	The instance identifier of the message being acknowledged. The default is the instance number of the last message to be received.

Name	**Usage**	**Description**
rp_name	Optional	The routing point to acknowledge on: ▪ SEWS_to_egate ▪ SEWS_from_egate The default depends on the direction at authentication: SEWS_to_egate if the direction is "to_egate", and SEWS_from_egate otherwise
force	Optional (value = **y**)	Use this parameter, a value of **y**, to acknowledge a message that was *not* the last one received with a GET request.

Returns

Success:

Value	**Description**
s_umid	The unique identifier of the message that has been acknowledged.
instance	The instance identifier of the message that has been acknowledged.

Failure:

Value	**Type**	**Description**
code	number (**nnn**)	The following codes can be returned: **502**, **504**, **505**, **507**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

Location

```
\monk_library\swiftadk\adk-ack.monk
```

adk-connect

Description

Once connected to the remote system, call this routine to connect to SEWS. The routine performs the following:

1. Creates inbound and outbound connections to SEWS.
2. Creates a transaction log file for each direction.
3. Initiates Event recovery if existing log file is found.
4. Reads the challenge supplied by the SEWS **CHAL**.
5. Calculates and sends the appropriate response.
6. Checks the reply to make sure the **AUTH** has been successful.
7. Get a list of reserved messages on the routing point for this connection.
8. If a reserved message is found, the function starts an Event recovery routine. An error during this procedure causes the e*Way to abort and wait for manual recovery.

Signature

```
(adk-connect)
```

Parameters

None.

Returns

Returns true (#t) upon success or false (#f) upon failure.

Throws

Exception	Content
adk-exception-auth	"Unable to authenticate from_egate"
	"Unable to authenticate to_egate"
adk-exception-journal	"Journal failed: "
adk-exception-recover	"Messages outstanding on routing point"
	"Recovery failed (from egate)"
	"Recovery failed (to egate)"

Additional Information

- Each connection is either **SEWS_to_egate** or **SEWS_from_egate**. The **LIST** operation gets a list of **s_umids** only for the associated SWIFTAlliance routing point.
- Event recovery results are entered into the SWIFTAlliance journal log.

Location

```
\monk_library\swiftadk\adk-connect.monk
```

adk-disconnect

Description

Disconnects the e*Way from SEWS.

Signature

```
(adk-disconnect)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-disconnect.monk
```

adk-incoming

Description

This function asks for an available (unreserved) message from the SWIFT routing point. If there is one, **adk-incoming** writes it to the queue and sends an **ACK** to SWIFT.

Signature

```
(adk-incoming)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-incoming.monk
```

adk-init

Description

Initializes system by:

1. Loading Monk extensions.
2. Setting up global variables.

3. Defining internal exceptions:
    - adk-exception-auth
    - adk-exception-recover

Signature

```
(adk-init)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-init.monk
```

adk-init-inbound

Description

Initialization function for inbound (SWIFT to e*Gate) operation. Identical to **adk-init** except that definition for inbound direction included in global variables.

Signature

```
(adk-init-inbound)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-init-inbound.monk
```

adk-init-outbound

Description

Initialization function for outbound (e*Gate to SWIFT) operation. Identical to **adk-init** except that definition for outbound direction included in global variables.

Signature

```
(adk-init-outbound)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-init-outbound.monk
```

adk-nak

Description

Sends a negative acknowledgment to SEWS that the last message received from a **GET** request could *not* be successfully committed to disk.

Signature

```
(adk-nak s_umid instance rp_name force)
```

Parameters

Name	Usage	Description
s_umid	Mandatory	The unique identifier of the message being acknowledged
instance	Optional	The instance identifier of the message being acknowledged. The default is the instance number of the last message to be received.
rp_name	Optional	The routing point to acknowledge on: ▪ SEWS_to_egate ▪ SEWS_from_egate The default depends on the direction at authentication: SEWS_to_egate if the direction is "to_egate", and SEWS_from_egate otherwise
force	Optional (value = **y**)	Use this parameter, a value of **y**, to acknowledge a message that was *not* the last one received with a GET request.

Returns

The value returned depends on the success or failure of the **NAK**.

Success:

Value	Description
s_umid	The unique identifier of the message that has been acknowledged.

Value	Description
instance	The instance identifier of the message that has been acknowledged.

Failure:

Value	Type	Description
code	number (**nnn**)	The following codes can be returned: **502**, **504**, **505**, **507**, and **517**. Content contains a description of the error. See **Error Codes** on page 32.

Throws

None.

Location

```
\monk_library\swiftadk\adk-nak.monk
```

adk-outgoing

Description

This function takes a SWIFT message as a string, and sends it to SEWS.

Signature

```
(adk-outgoing)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-outgoing.monk
```

adk-shutdown

Description

Default shutdown function; deletes transaction log file, sends Journal shutdown request to SWIFT.

Signature

```
(adk-shutdown)
```

Parameters

None.

Returns

None.

Throws

Exception	Content
adk-exception-journal	"Journal failed: "

Location

```
\monk_library\swiftadk\adk-shutdown.monk
```

adk-startup

Description

Default startup function.

Signature

```
(adk-startup)
```

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-startup.monk
```

adk-verify

Description

Default verification function.

Signature

(adk-verify)

Parameters

None.

Returns

None.

Throws

None.

Location

```
\monk_library\swiftadk\adk-verify.monk
```

8.5 Generic e*Way Functions

The functions described in this section are implemented in the e*Way Kernel layer and control the e*Way's most basic operations. They can be used only by the functions defined within the e*Way's configuration file. None of these functions is available to Collaboration Rules scripts executed by the e*Way. These functions are located in **stcewgenericmonk.exe**.

The current set of basic Monk functions is:

event-commit-to-egate on page 123

event-rollback-to-egate on page 124

event-send-to-egate on page 124

event-send-to-egate-ignore-shutdown on page 125

event-send-to-egate-no-commit on page 125

get-logical-name on page 126

insert-exchange-data-event on page 126

send-external-up on page 127

send-external-down on page 127

shutdown-request on page 128

start-schedule on page 128

stop-schedule on page 129

waiting-to-shutdown on page 129

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 125

event-send-to-egate-no-commit on page 125

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 124

event-send-to-egate-no-commit on page 125

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 123

event-rollback-to-egate on page 124

event-send-to-egate on page 124

event-send-to-egate-ignore-shutdown on page 125

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 80

Zero Wait Between Successful Exchanges on page 81

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