SeeBeyond ICAN Suite

e*Way Intelligent Adapter for SNA User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

SEEBEYOND®

Version 20050406042415.

Contents

Preface 7

Intended Reader 7

Organization 7

Nomenclature 8

Online Use 8

Writing Conventions 8

Additional Documentation 9

Chapter 1

Introduction 10

SNA Architectural Overview 10

Supported Logical Unit Types 13

SNA LU6.2 13

SNA LUA 14

SNA LU0 14

SNA e*Way Overview 15

e*Way Components 15

Supported Operating Systems 16

Chapter 2

Installation 17

System Requirements 17

Supported Operating Systems 17

English Version 17

Korean Version 18

Environment Configuration 18

External System Requirements 19

SNA LU6.2 19

SNA LU0, LU1, LU2, LU3 19

Solaris Patch Requirements 19

External Configuration Requirements 20
Configuring the SNA Server and Client 20
All Platforms 20
Additional Procedures for Solaris 20
Installing the e*Way 21
Windows Systems 21
Installation Procedure 21
Subdirectories and Files 23
UNIX Systems 24
Installation Procedure 24
Subdirectories and Files 25
Optional Example Files 26
Installation Procedure 26
Subdirectories and Files 27
Chapter 3
Implementation 29
Overview 29
Pre-Implementation Tasks 29
Implementation Sequence 30
Viewing e*Gate Components 30
Creating a Schema 31
Creating Event Types 32
Creating Event Type Definitions 32
Assigning ETDs to Event Types 32
Defining Collaborations 34
Creating Intelligent Queues 35
Exception Handling 36
Enabling TP Trace 37
Known Issues and Limitations 37
Sample Schema 38
LU6.2 38
Chapter 4
Setup Procedures 42
Overview 42
Setting Up the e*Way 43
Creating the e*Way 43
Modifying e*Way Properties 44
Configuring the e*Way 45
Using the e*Way Editor 46

Changing the User Name 49
Setting Startup Options or Schedules 49
Activating or Modifying Logging Options 51
Activating or Modifying Monitoring Thresholds 52

Troubleshooting the e*Way 53
Configuration Problems 53
System-related Problems 54

Chapter 5

Operational Overview 55

e*Way Architecture 55

Basic e*Way Processes 57
Initialization Process 58
Connect to External Process 59
Data Exchange Process 60
Disconnect from External Process 63
Shutdown Process 63

Chapter 6

Configuration Parameters (LU6.2) 64

Overview 64

General Settings 65

Communication Setup 67

Monk Configuration 70
Specifying Function or File Names 70
Specifying Multiple Directories 70
Load Path 70

SNA Client Configuration 79

Chapter 7

Configuration Parameters (LUA) 82

Overview 82

General Settings 83

Communication Setup 85

Monk Configuration 88
Specifying Function or File Names 88
Specifying Multiple Directories 88
Load Path 88

SNA LUA Client Configuration 97

Chapter 8

API Functions 98

Overview 98

Native e*Way Functions 99

LU6.2 99

LUA 108

Standard e*Way Functions 113

LU6.2 113

LUA 120

Generic e*Way Functions 126

Index 134

Preface

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

P.1 Intended Reader

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

- Windows and UNIX operations and administration
- Windows-style GUI operations
- SNA Server, LU6.2 and/or LU0, and CPIC APIs

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 5-8, 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 SNA is frequently referred to as the SNA e*Way, or simply the e*Way.

P.4 Online Use

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

Hyperlinking is available in:

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

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

P.5 Writing Conventions

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

Monospaced (Courier) Font

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

```
Configuration for BOB_Promotion

java -jar ValidationBuilder.jar
```

Variables within a command line, 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

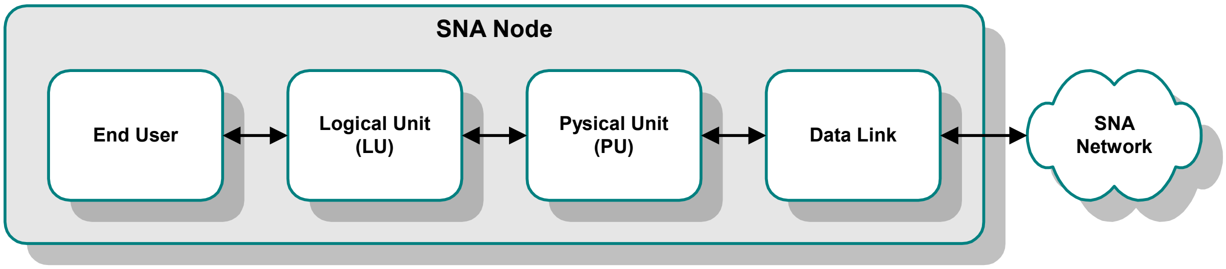
This chapter provides a brief overview on SNA fundamentals and an introduction to the e*Way Intelligent Adapter for SNA.

1.1 SNA Architectural Overview

SNA (System Network Architecture) is a data communications architecture developed by IBM to specify common conventions for communication between various IBM hardware and software products. It is specifically designed to address issues of the reliability and flexibility of sharing data between components and their peripherals. Many vendors other than IBM also support SNA, allowing their products to interact with SNA networks.

An addressable unit on an SNA network is called a node, and is made up of four functional components forming a hierarchy as shown in Figure 1.

Figure 1 SNA Node Architecture

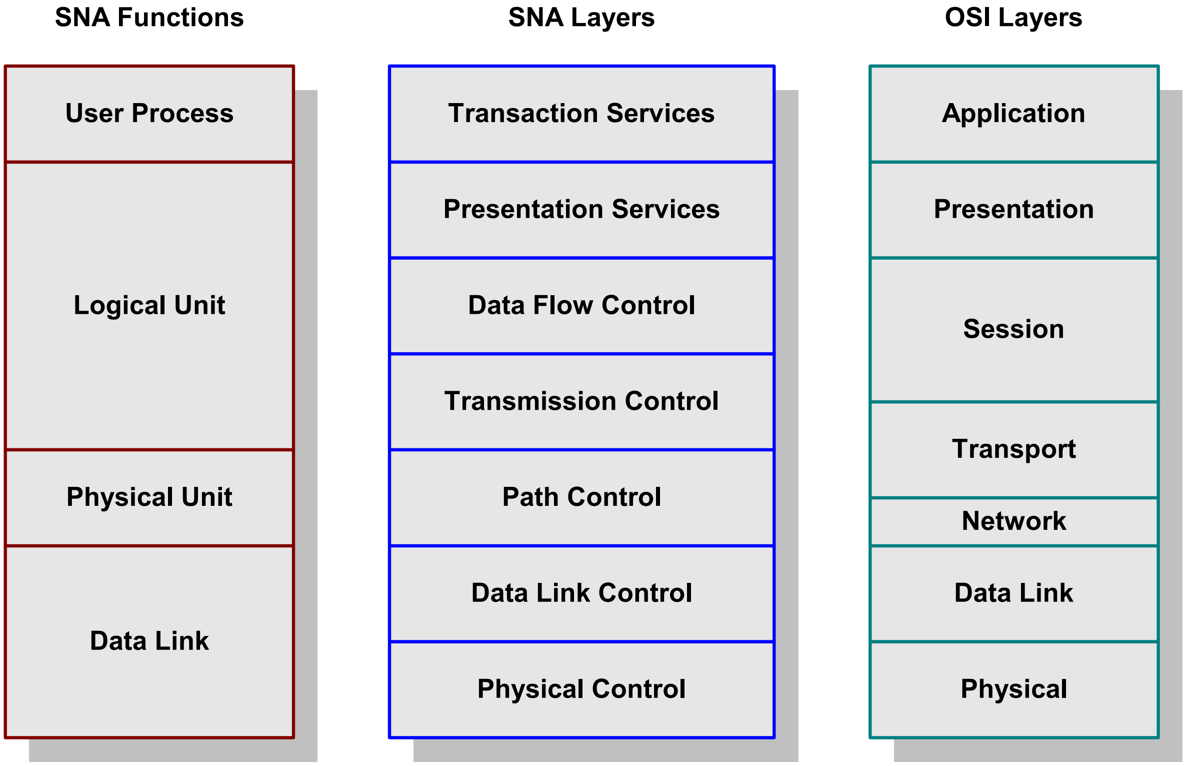


To establish a communications session, SNA uses Logical Units (LUs) as entry points into the network. There are several types of LUs, currently type 0 through type 6.2. Most of the LU types are specific to IBM operating environments, but type 6 is intended for use in a distributed data processing environment.

Generally, an LU can communicate only with another LU of the same type, but specific exceptions to this rule exist with type 6.2. LU6.2 is the least-restrictive of the various LU types, and also supports multiple concurrent sessions. As a result, it is the LU most widely supported by other system vendors.

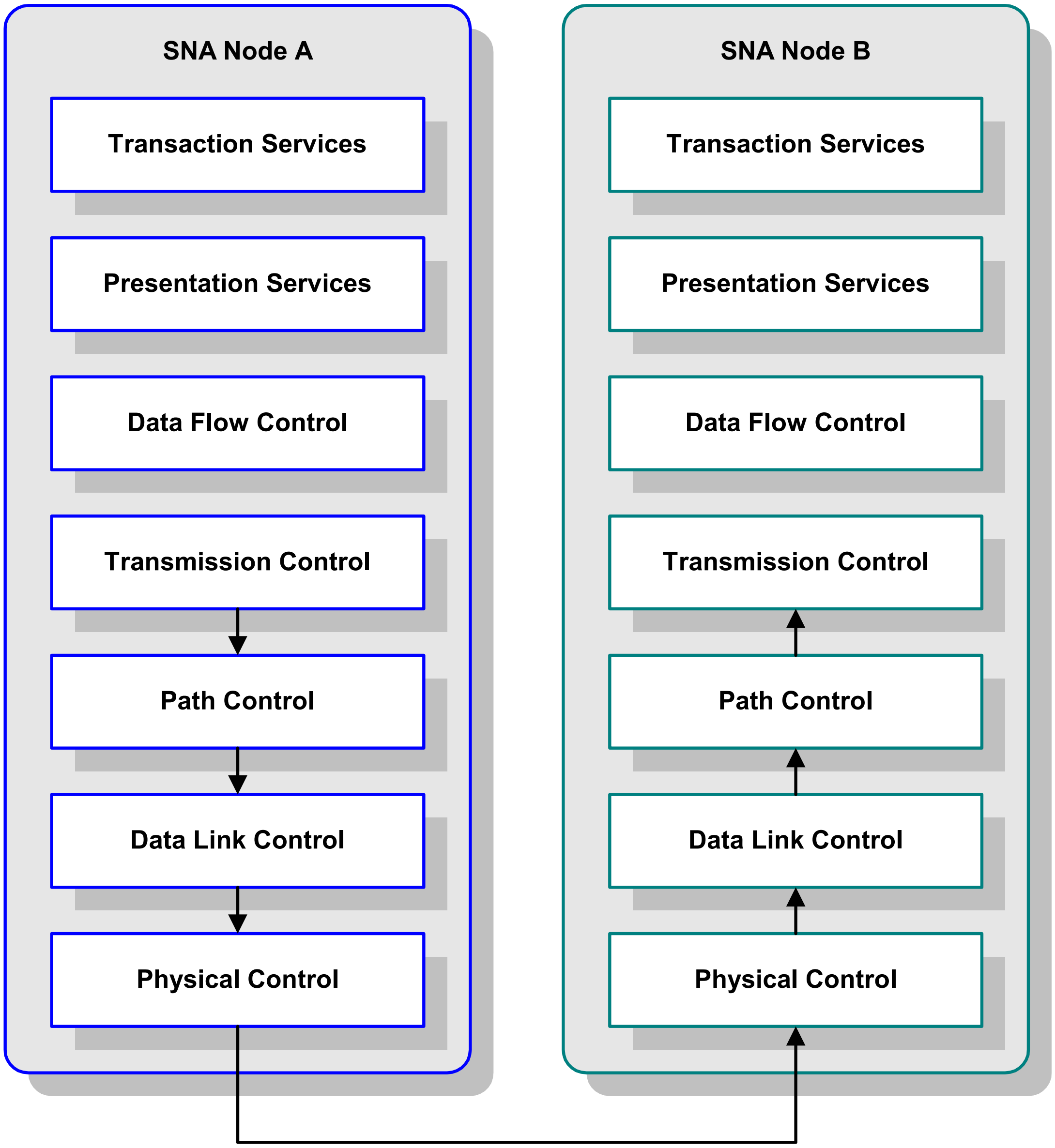
Like the OSI model, SNA functions are divided into seven hierarchical layers, but the layers are not identical. Their relationships to each other, and to the SNA node functionality, are shown in Figure 2. The Transport Network handles the lower three layers, while the Network Accessible Units (NAU) implement the upper four layers by using the services of the Transport Network to establish communication between nodes.

Figure 2 SNA Functional Layers



SNA defines formats and protocols between these layers that allow equivalent layers in different nodes to communicate with each other. Also, each layer provides services to the layer above, and requests services from the layer below. As an example, the communication path between two Transmission Control layers would appear as shown in Figure 3.

Figure 3 Equivalent-Layer Communications Path

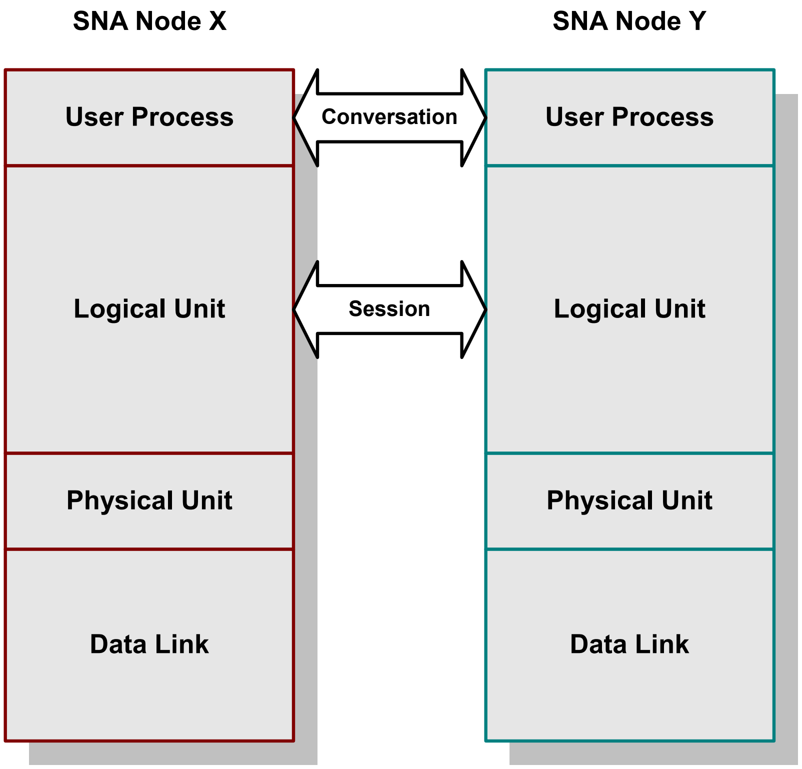


SNA uses a standard method for the exchange of data within a network. This standard method defines how to establish a route between components, how to send and receive data reliably, how to recover errors, and how to prevent flow problems.

Originally designed for networks in which a mainframe computer controls the communications relationships, SNA has since evolved to incorporate protocols and implementations to allow two user processes to communicate with each other directly. These two different networking models, or roles, are referred to as hierarchical and peer-oriented, respectively. The peer-oriented model is designed to allow distributed control of the communications process independent of the mainframe.

The peer-to-peer connection between two user processes is known as a *conversation*, while the peer-to-peer connection between two LUs is known as a *session*. A session is generally a long-term connection between two LUs, while a conversation is generally of shorter duration.

Figure 4 Sessions and Conversations



What is shown in Figure 2 and Figure 4 as a *User Process* is also known as a *Transaction Program* (TP). Also, the interface between a User Process and an LU is known as *Presentation Services*.

1.1.1 Supported Logical Unit Types

SNA LU6.2

LU 6.2, also known as APPC (Advanced Program-to-Program Communication), is used for Transaction Programs communicating with each other in a distributed data processing environment. In a CPIC (Common Programming Interface for Communications) implementation, CPIC provides the API that contains the commands, known as verbs, that are used by LU 6.2 to establish communication sessions.

Two types of Presentation Service interfaces are possible with LU6.2: mapped conversations and unmapped, or basic, conversations. Table 1 summarizes the set of LU6.2 commands for basic conversations. Equivalent commands for mapped conversations have the prefix <**MC_**> added to the command name. Note that “control operator verbs” are not listed.

Table 1 LU6.2 Commands

Name	Description
ALLOCATE	Allocates a conversation with another program.
CONFIRM	Sends a confirmation request to the remote process and waits for a reply.
CONFIRMED	Sends a confirmation reply to the remote process.
DEALLOCATE	De-allocates a conversation.
FLUSH	Forces the transmission of the local SEND buffer to the other LU.
GET_ATTRIBUTES	Obtains information about a conversation.
PREPARE_TO_RECEIVE	Changes the conversation state from SEND to RECEIVE.
RECEIVE_AND_WAIT	Waits for information (either data or confirmation request) to be received from the partner process.
RECEIVE_IMMEDIATE	Receives any information that is available in the local LU's buffer, but does not wait for information to arrive.
REQUEST_TO_SEND	Notifies the partner process that the local process wants to send data. When a "send" indication is received from the partner process, the conversation state changes.
SEND_DATA	Sends one data record to the partner process.
SEND_ERROR	Informs the partner process that the local process has detected an application error.

SNA LUA

The e*Way Intelligent Adapter for SNA uses the Conventional Logical Unit Application (LUA) interface from Data Connection Limited to communicate with LU0, LU1, LU2, and LU3 hosts, using their SNAP-IX SNA function library. The LUA interface acts at the request/response unit (RU) level, and supports an extensive set of functions.

SNA LU0

The e*Way Intelligent Adapter for SNA also supports the LU0 interface from Data Connection Limited to communicate with LU0 hosts. This provides a less-complex, alternative interface that supports a subset of the functions contained in the LUA interface.

Note: *The e*Way Intelligent Adapter for SNA currently supports LUA and LU0 only on the Solaris operating system.*

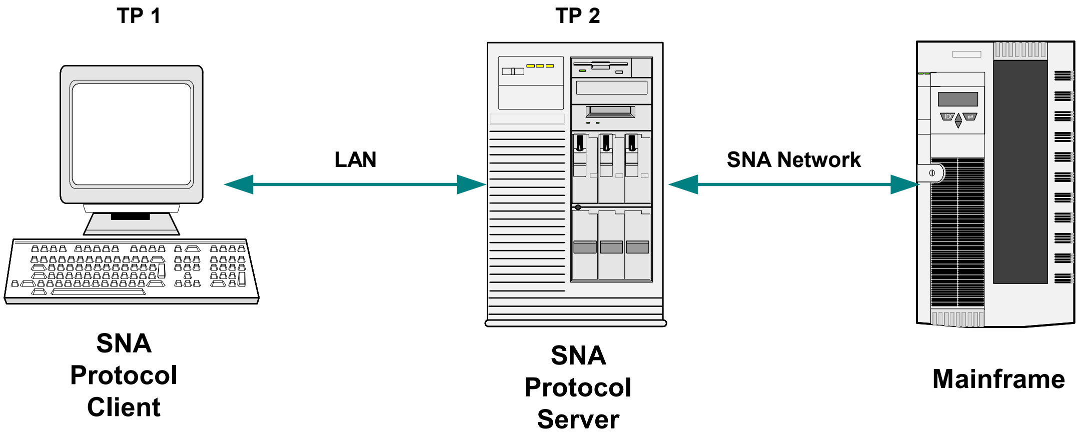
1.2 SNA e*Way Overview

The SNA e*Way is an interface that makes uni-directional calls to an SNA Server. The SNA Server acts as a high-speed gateway between distributed SNA Clients and the SNA network having a mainframe host system (see Figure 5).

The SNA e*Way enables the SeeBeyond e*Gate Integrator system to access an SNA network environment to drive entire transactions, including conversational transactions. The connection requires a TCP/IP connection with, and the appropriate link service to, the SNA server in use. The SNA Client and the e*Gate Participating Host reside on the same platform.

In a typical data exchange using the SNA e*Way, the e*Way invokes either the LU6.2 or LU0 protocol to enable the SNA client to send requests to the SNA server.

Figure 5 SNA Data Exchange



Note: *The SNA e*Way does not support bi-directional transaction calls. Two e*Ways must be configured to handle inbound and outbound data transfer.*

1.2.1 e*Way Components

The SNA e*Way incorporates the following components:

- **stcewgenericmonk.exe**, the executable component (installed with e*Gate)
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk function scripts, discussed in **Chapter 8**.

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

1.2.2 Supported Operating Systems

The e*Way Intelligent Adapter for e*Gate currently supports the following combinations of LU types and operating systems.

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

Table 2 English-language Version

Operating System	LU0	LU1	LU2	LU3	LU6.2
Windows 2000, Windows XP, and Windows Server 2003	-	-	-	-	X
IBM AIX 5.1L and 5.2	-	-	-	-	X
Sun Solaris 8	X	X	X	X	X

Table 3 Korean-language Version

Operating System	LU0	LU1	LU2	LU3	LU6.2
Sun Solaris 8	X	X	X	X	X

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

2.1 System Requirements

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

1. An e*Gate Participating Host.
2. A TCP/IP network connection.
3. Sufficient free disk space to accommodate e*Way files:
   - Approximately 200 KB on Windows systems
   - Approximately 400 KB on Solaris systems
   - Approximately 200 KB 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.*

2.1.1 Supported Operating Systems

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

English Version

English versions of:

- Windows 2000, Windows 2000 SP1, Windows 2000 SP2
- Windows XP
- Solaris 8

- Solaris 9
- AIX 5.1
- AIX 5.2

Korean Version

Korean versions of:

- Solaris 8
- Solaris 9

2.1.2 Environment Configuration

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

2.2 External System Requirements

2.2.1 SNA LU6.2

To enable the e*Way to communicate properly with the SNA Server system, the following are required:

- Microsoft e*Gate Server 4.0 client
- Administrative access to the SNA server
  - Sunlink e*Gate Server 9.1 (Solaris)
  - IBM Communication Server 6.0 (AIX)
- CPI-C version 1.2
- Appropriate link service for the e*Gate Server in use

2.2.2 SNA LU0, LU1, LU2, LU3

Data Connection Limited's SNAP-IX library is required for the e*Way to communicate properly with an e*Gate LU0, LU1, LU2, or LU3 Server system.

2.2.3 Solaris Patch Requirements

Solaris operating systems require the following e*Gate version 9.1 patches before the e*Gate e*Way can be installed. If the patch is not installed, the setup program detects it. These patches can be downloaded from **http://sunsolve.sun.com**.

Table 4 Sun-Solaris Patches

Package	SNA component	Patch
SUNWpu21	pu21server	106162-29
SUNWgman	gateway mngr	106164-15
SUNWgmi	configuration gui	106165-09
SUNWlu62	lu62 configs	105860-23

Note: *Once these patches have been installed, the configuration file shows two* **pu2***s. Use* ***vi*** *to edit out one of them. Each time the configuration is changed, you must start up the sunsetup script:*

```
<fullpath>/opt/SUNWpu21/.sunsetup
```

The **sunsetup** menu provides a list of options.

1. Select Option 6 (stop **pu21**).
2. Select Option 7 (stop **gman**).
3. Select Option 4 (start **gman**).

Note: *When bringing down the SNA server, you must invoke option 6 and 7, but in bringing up the SNA server, you must invoke option 4. The* **gman** *automatically brings up your active SNA configuration.*

2.3 External Configuration Requirements

Note: *The configuration steps mentioned below are presented as a general guideline for configuring the SNA system, and are not to be considered complete. Please refer to your SNA Administration guide for detailed information on SNA System Configuration. Each platform requires different parameters and information.*

2.3.1 Configuring the SNA Server and Client

All Platforms

You must configure both the partner and the remote SNA systems to have an active connection. Use the following procedure as a guide.

1. Configure a link station or service for the remote and partner system. This can be an Ethernet or Token ring link for the LAN connection. Links vary for SDLC, QLLC and channel connections.
2. Configure a local LU and a remote LU definition on each system. You need the physical machine address, the control point name, or full computer name, and the network name.
3. Define a mode on the remote and local SNA system. This mode name must be the same on both systems in order to have an active connection and for the data to be transferred.
4. Define a Symbolic Destination Name and Transaction Program (TP) name on both systems. The names must match in order for the TPs to communicate with each other. You must select the correct mode name for each TP name.
5. The status flag for **DEALLOCATE** must be a **4**. You must set your external system to send or receive this flag for the e*Way to process a shutdown.

Additional Procedures for Solaris

1. Create an information file that the e*Way can access. This file should have the same name as entered for the e*Way's **SYMDESTNAME** parameter.
2. Set the appropriate environmental variables for **APPC_GATEWAY** and **APPC_LOCA_LU**.

2.4 Installing the e*Way

2.4.1 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*).

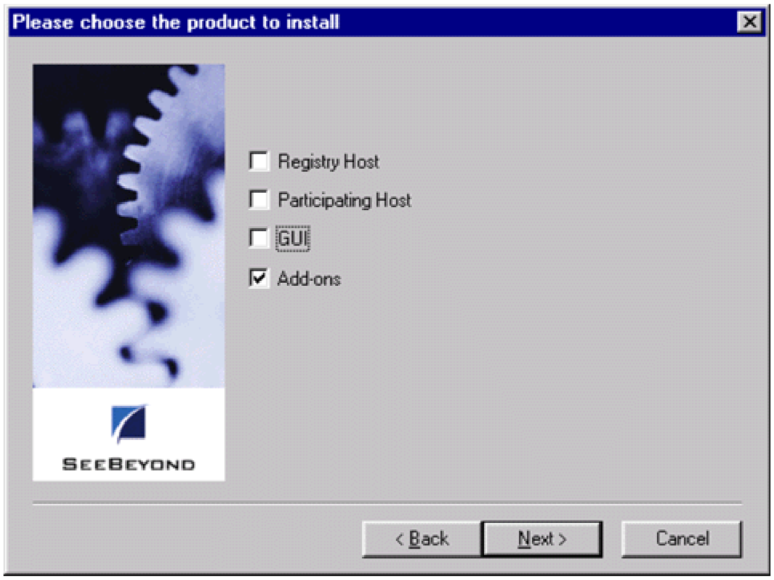
2 Exit all Windows programs and disable any anti-virus applications before running the setup program.

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

Figure 6 Choose Product Dialog



Please choose the product to install

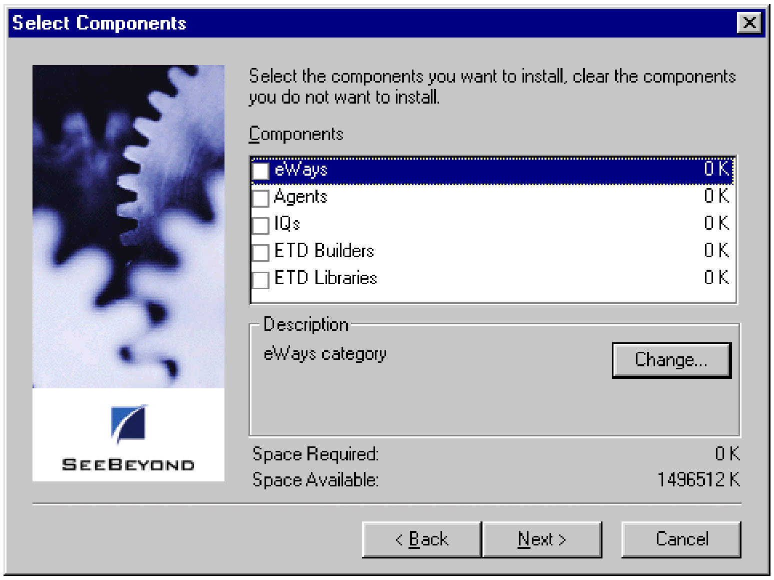
□ Registry Host
□ Participating Host
□ GUI
☑ Add-ons

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 7). Highlight—*but do not check*—**eWays** and then click **Change**.

Figure 7 Select Components Dialog



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

Components

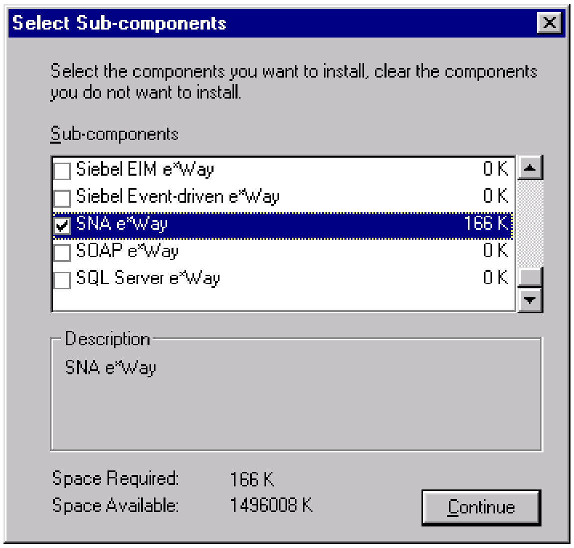
□ eWays	0 K
□ Agents	0 K
□ IQs	0 K
□ ETD Builders	0 K
□ ETD Libraries	0 K

Description
eWays category

Space Required: 0 K
Space Available: 1496512 K

6 When the **Select Sub-components** dialog box appears (see Figure 8), check the **e*Gate e*Way**.

Figure 8 Select e*Way Dialog



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

Sub-components

□ Siebel EIM e*Way	0 K
□ Siebel Event-driven e*Way	0 K
☑ SNA e*Way	166 K
□ SOAP e*Way	0 K
□ SQL Server e*Way	0 K

Description
SNA e*Way

Space Required: 166 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 5 Participating Host & Registry Host

Subdirectories	Files
\bin\	stc_monksna.dll
\configs\stcewgenericmonk\	stcewsna.def stcewsnalu0.def
\monk_library\	ewsna.gui
\monk_library\ewsna\	sna-conn-establish.monk sna-conn-shutdown.monk sna-conn-verify.monk sna-incoming.monk sna-init.monk sna-neg-ack.monk sna-outgoing.monk sna-pos-ack.monk san-shutdown.monk sna-startup.monk

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

Table 6 Registry Host Only

Subdirectories	Files
\	stcewsna.ctl

2.4.2 UNIX Systems

Installation Procedure

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

To install the e*Way on a UNIX system

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

```
cd /cdrom
```

4. Start the installation script by typing:

```
setup.sh
```

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

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

Subdirectories and Files

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

Table 7 Participating Host & Registry Host

Subdirectories	Files
/bin/	stc_monksna.dll
/configs/stcewgenericmonk/	stcewsna.def stcewsnalu0.def
/monk_library/	ewsna.gui
/monk_library/ewsna/	sna-conn-establish.monk sna-conn-shutdown.monk sna-conn-verify.monk sna-incoming.monk sna-init.monk sna-neg-ack.monk sna-outgoing.monk sna-pos-ack.monk san-shutdown.monk sna-startup.monk

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

Table 8 Registry Host Only

Subdirectories	Files
/	stcewsna.ctl

2.5 Optional Example Files

The installation CD-ROM contains a pair of sample schema located in the **samples\ewsna** directory.

- **sna_client.zip**

  This Client schema receives a file (**testdata.fin**) and uses a local SNA e*Way to send it to a remote SNA e*Way. It then receives this file back through its inbound SNA e*Way from another remote SNA e*Way and writes the Event to a file.

- **sna_server.zip**

  This is the remote schema that receives the message sent from the Client schema, and then routes the message back to the SNA client.

- **testdata.fin**

  This is the input data for the Client schema.

To use the schemas, you must load them onto your system using the following procedure. See **Sample Schema** on page 38 for descriptions of the sample schema and instructions regarding its use.

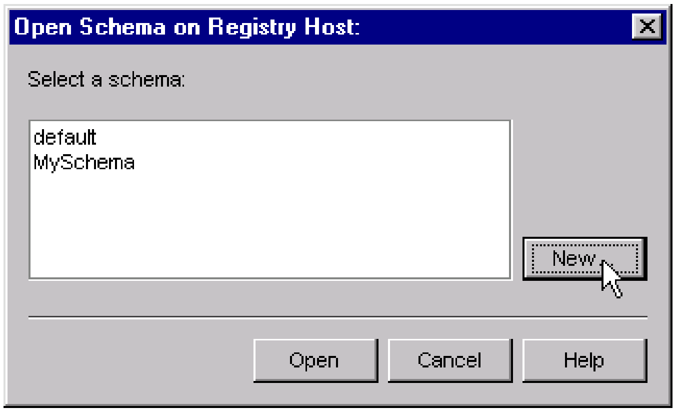
Note: *The e*Gate e*Way must be properly installed on your system before you can run the sample schema.*

2.5.1 Installation Procedure

To load a sample schema

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

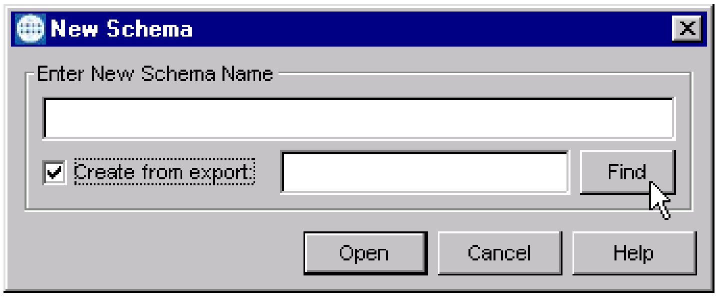
Figure 9 Open Schema Dialog



2 Type the name you want to give to the schema (for example, **SNA.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 the SNA Client schema (**sna_client.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* **sna_client.zip** *to a local directory and extract the files. Using a text editor, edit the file* **sna_client.exp***, replacing all instances of the name* **localhost** *with your desired name. Add the edited* **.exp** *file back into the* **.zip** *file.*

5 Create a directory **\eGate\data\input** and copy the file **testdata.fin** into that directory.

6 Create a second directory **\eGate\data\sna\output** for the output data files.

7 On another platform running SNA, import the SNA Server schema (**sna_server.zip**) following the same procedure as in steps 1 - 4.

8 Create a directory.**\eGate\data\sna\output** for the output data files.

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 9 Subdirectories and Files - Server Schema

Subdirectories	Files
\	server.ctl
\runtime\configs\stcewfile\	eater.cfg eater.sc
\runtime\configs\stcewgenericmonk\	SNAInbound.cfg SNAInbound.sc SNAOutbound.cfg SNAOutbound.sc
\runtime\monk_scripts\common\	blob.ssc

Table 10 Subdirectories and Files - Client Schema

Subdirectories	Files
\	client.ctl
\runtime\configs\stcewfile\	eater.cfg eater.sc
\runtime\configs\stcewgenericmonk\	feeder.cfg feeder.sc SNAInbound.cfg SNAInbound.sc SNAOutbound.cfg SNAOutbound.sc
\runtime\monk_scripts\common\	blob.ssc genericmonk-connect.monk genericmonk-eater.monk genericmonk-feeder.monk genericmonk-finish.monk genericmonk-on-ack.monk genericmonk-on-nak.monk genericmonk-startup.monk

Chapter 3

Implementation

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

3.1 Overview

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

Note: *The SNA e*Way does not support bi-directional transactions. Two e*Ways must be configured to handle inbound and outbound data transfer.*

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

3.1.1 Pre-Implementation Tasks

Installation of the e*Way

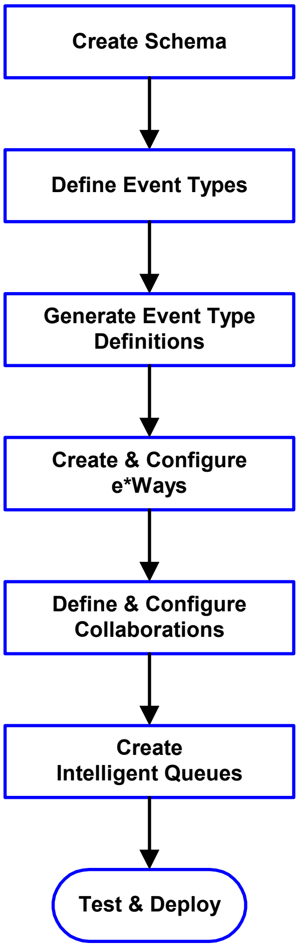
The first task, of course, is to install the e*Way as described in **Chapter 2**.

Installation of Sample Schemas

If you want to make use of the provided sample schemas, you must install them manually as described in **Optional Example Files** on page 26.

Note: *It is highly recommended that you make use of the sample schemas to familiarize yourself with e*Way operation, test your system, and use as templates for your working schemas.*

3.1.2 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 31).

2 The second step is to define the Event Types you are transporting and processing within the Schema (see **Creating Event Types** on page 32).

3 Third, you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) derived from the applicable Business Rules (see **Creating Event Type Definitions** on page 32).

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

5 Next is to define and configure the Collaborations linking the Event Types from step 2 (see **Defining Collaborations** on page 34).

6 Now you need to create Intelligent Queues to hold published Events (see **Creating Intelligent Queues** on page 35

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

3.1.3 Viewing e*Gate Components

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

3.2 Creating a Schema

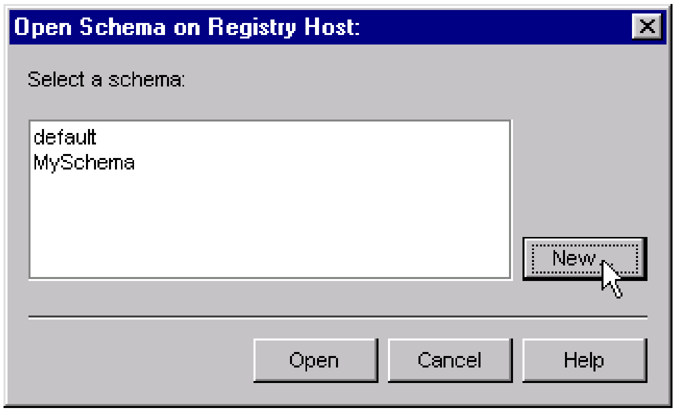
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

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

To select or create a schema

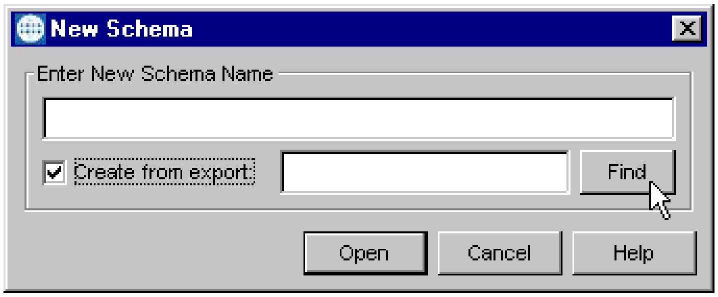
1 Invoke the **Open Schema** dialog box and **Open** an existing schema or click **New** to create a new schema.

Figure 11 Open Schema Dialog



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 Schema Designer then opens under your new schema name.

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

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

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

3.3 Creating Event Types

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

To define the Event Types

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

3.4 Creating Event Type Definitions

Each Event Type now must be associated with an Event Type Definition within the schema. In general, you select an existing ETD or create a new one based on an existing template. See the *e*Gate Integrator User's Guide* for additional information.

To create an Event Type Definition

1. In the e*Gate Event Type Editor, select **Build**.
2. In the **Build an Event Type Definition** dialog box, locate and select an ETD to use as a template.
3. Edit the ETD properties as needed.
4. Rename and save as a new ETD (**.ssc** file).

3.4.1 Assigning ETDs to Event Types

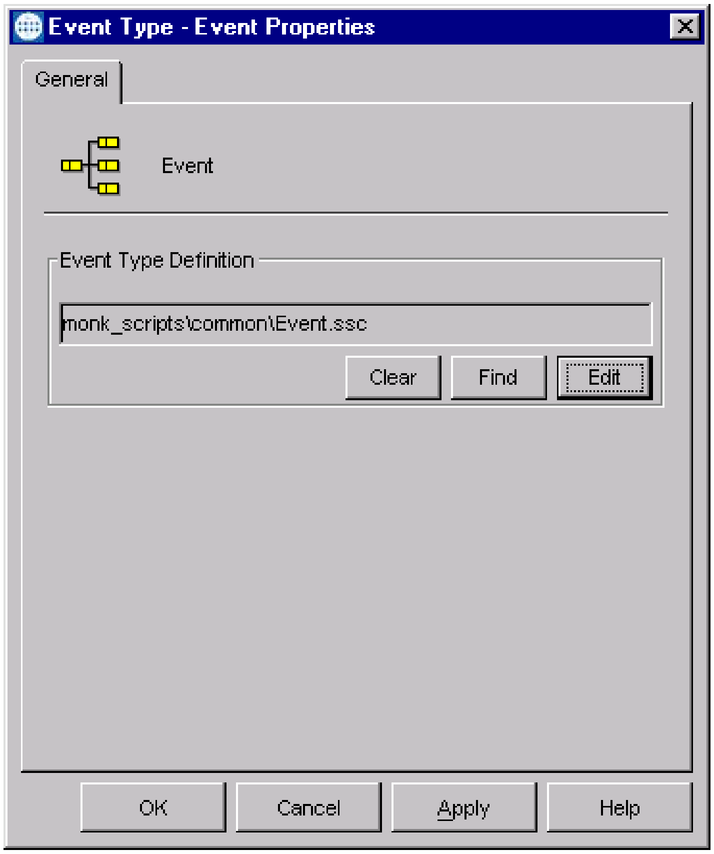
After you have created the e*Gate system's ETD files, you can assign them to existing Event Types.

To assign ETDs to Event Types

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

   The Event Type Properties dialog box appears. See Figure 13.

Figure 13 Event Type Properties Dialog Box



Event Type Definition

monk_scripts\common\Event.ssc

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

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

6 Click **Select**. The file populates the Event Type Definition field.

7 To save any work in the properties dialog box, click **Apply** to enter it into the system.

8 When finished assigning ETDs to Event Types, click **OK** to close the properties dialog box and apply all the properties.

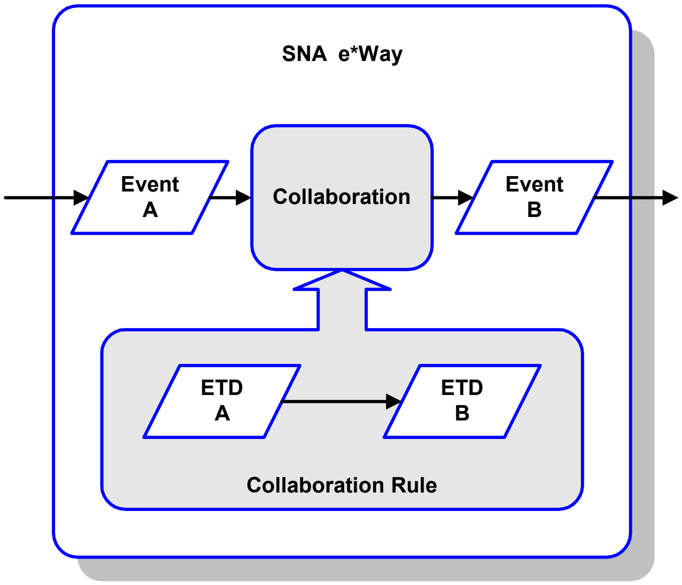
Each Event Type is now associated with the specified Event Type Definition.

3.5 Defining Collaborations

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

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

Figure 14 Collaborations



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

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

3.6 Creating Intelligent Queues

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.

3.7 Exception Handling

The SNA e*Way handles an external (remote) shutdown request by confirming the request and throwing an application-specific exception. Specifically, if the remote application issues a **deallocate**, the e*Way then throws the exception **$Sna-Exception-Fatal** back to the calling Monk function. Please refer to the *Exception Functionality* chapter of the *Monk Developers Reference* for details on catching exceptions.

Example Code

The following code sample, from the monk script **sna_incoming.monk**, demonstrates how to catch this exception and issue a shutdown request to shut the e*Way down.

```
(if (string=? SNA_CONFIGURATION_SYNCHRONIZATION_LEVEL "NONE")
    (begin
     (try
       (set! pszData (sna-client-recv-no-synch hCon
SNA_CONFIGURATION_PACKETSIZE SNA_CONFIGURATION_TIMEOUT))
       (catch
         (($Sna-Exception-Fatal)
           (display (string-append "Exception string: "
                    (exception-string) "."))
           (newline)
           (display "Caught Fatal Exception - calling shutdown\n")
           (shutdown-request)
         )
         (otherwise
           (display (string-append "Exception category: "
                    (number->string (exception-category)) "."))
           (newline)
           (display (string-append "Exception symbol: "
                    (symbol->string (exception-symbol)) "."))
           (newline)
           (display (string-append "Exception string: "
                    (exception-string) "."))
           (newline)
         )
       ); catch
     ); try
    ); begin
```

3.8 Enabling TP Trace

On Solaris only, SNA LU6.2 TP trace can be turned on by setting the following environment variable (if in C shell) prior to starting the e*Way:

```
setenv SUNLINK_CNT_API_TRACE 1
export SUNLINK_CNT_API_TRACE
```

A TP trace is written to the current directory.

3.9 Known Issues and Limitations

1. SNA e*Ways that send initialization must be started after the accepting program is ready to accept.
2. The status flag for deallocate must be a **4**. You must set your external system to send or receive this flag for the e*Way to process a shutdown.
3. Issuing a shutdown while running in Non-Confirmed mode shuts down only the e*Way to which you issued the command. Issuing a shutdown while running in Confirmed mode shuts down the e*Way to which you issued the command and the associated e*Way.

3.10 Sample Schema

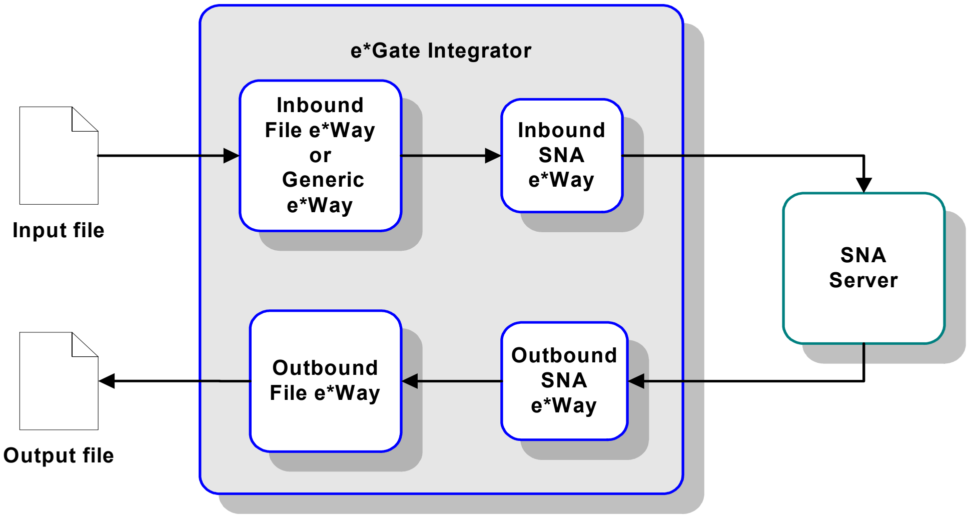
3.10.1 LU6.2

A sample schema pair for LU6.2 is provided with the e*Way to serve as an example for testing and template purposes. In the sample schemas, data is drawn from a text file using the file e*Way and sent to an external system using the SNA e*Way.

Note: *For testing purposes, the Inbound File e*Way is replaced by a Generic Monk e*Way, which enables repetitive processing.*

The data returned from the external system is received by the SNA e*Way, then forwarded to another file e*Way and stored in an output file on the local system (see Figure 15).

Figure 15 Sample Schema - Overview

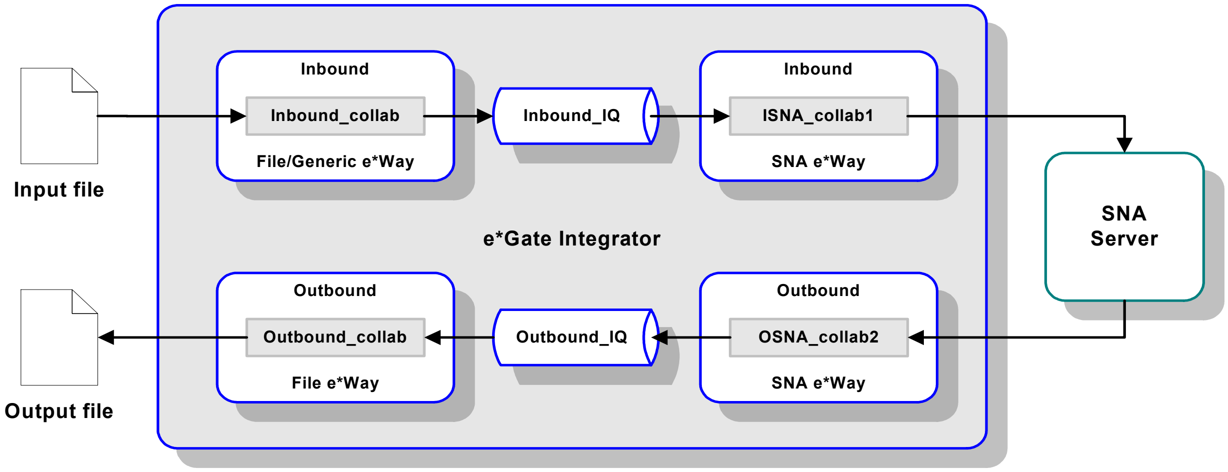


This Client schema incorporates several components, as listed in Table 11 and illustrated in **Figure 16** on page 39.

Table 11 Sample Schema Components

Type	Logical Name	Collaboration
SNA e*Way	inbound_sna_eway	ISNA_collab1
	outbound_sna_eway	OSNA_collab2
File e*Way	Inbound	Inbound_collab
	Outbound	Outbound_collab
Intelligent Queue	Inbound_IQ	N/A
	Outbound_IQ	N/A

Figure 16 Sample Schema - Detail



Collaboration Rule

The Collaboration Rule used by all Collaborations in the sample schema is named **Passthrough_Data**, and provides a Pass Through Collaboration service. It both subscribes and publishes to the Event Type Definition **In**.

Collaborations

Table 12 Inbound_collab Collaboration

Action	Event	Source/Destination
Subscribes to	In	<External>
Publishes to	In	Inbound_IQ

Table 13 ISNA_collab1 Collaboration

Action	Event	Source/Destination
Subscribes to	In	Inbound_collab
Publishes to	In	<External>

Table 14 OSNA_collab2 Collaboration

Action	Event	Source/Destination
Subscribes to	In	<External>
Publishes to	In	Outbound_IQ

Table 15 outbound_collab Collaboration

Action	Event	Source/Destination
Subscribes to	In	SNA_collab2
Publishes to	In	<External>

e*Way Configuration

Table 16 Inbound and Outbound SNA e*Ways

Section	Parameter	Value
General Settings	(all)	(default)
Communication Setup	Exchange Data Interval	0
	Zero Wait Between Successful Exchanges	No
Monk Configuration	Auxiliary Library Directories	monk_library/ewsna
	Monk Environment Initialization File	monk_library/ewsna/sna-init.monk
	Startup Function	sna-startup
	Process Outgoing Message Function	sna-outgoing
	Exchange Data With External Function	sna-incoming
	External Connection Establishment Function	sna-conn-establish
	External Connection Verification Function	sna-conn-verify
	External Connection Shutdown Function	sna-conn-shutdown
	Positive Acknowledgment Function	sna-pos-ack
	Negative Acknowledgment Function	sna-neg-nack
	(all others)	(blank)
SNA Client Configuration	SYMDESTNAME	*
	LocalTPName	*
	LOCAL_LU.Name	*
	PacketSize	*
	Timeout	*
	RequestReply	Yes
	Initialize Conversation	Yes or No, as appropriate
	Data Flow	Inbound or Outbound
	Synchronization Level	Confirm or None

* Enter the correct values for your system.

Table 17 Inbound File/Generic e*Way

Section	Parameter	Value
General Settings	AllowIncoming	Yes
	AllowOutgoing	No
Poller(inbound) Setting	Polldirectory	C:\TEMP (change if desired)
	Input File Mask	(default)

Table 18 Outbound File e*Way

Section	Parameter	Value
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
Poller(inbound) Setting	Polldirectory	C:\TEMP (change if desired)
	Input File Mask	sna_out.txt

Testing the Schema

1. Use the provided data file, **testdata.fin**, or create your own using any ASCII text editor.

2. Launch your schema. If the schema was configured properly and your connection to the test connection is valid, you should find response data from your requests in the file:

   ```
   \eGate\data\sna\output\sna_out.txt
   ```

Chapter 4

Setup Procedures

This chapter describes the procedures required to customize the SeeBeyond e*Way Intelligent Adapter for e*Gate to operate within your production system.

4.1 Overview

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

The topics discussed in this chapter include the following:

Setting Up the e*Way

Creating the e*Way on page 43

Modifying e*Way Properties on page 44

Configuring the e*Way on page 45

Changing the User Name on page 49

Setting Startup Options or Schedules on page 49

Activating or Modifying Logging Options on page 51

Activating or Modifying Monitoring Thresholds on page 52

Troubleshooting the e*Way

Configuration Problems on page 53

System-related Problems on page 54

4.2 Setting Up the e*Way

Note: *The SNA e*Way does not support bidirectional transactions. Two e*Ways must be configured to handle inbound and outbound data transfer.*

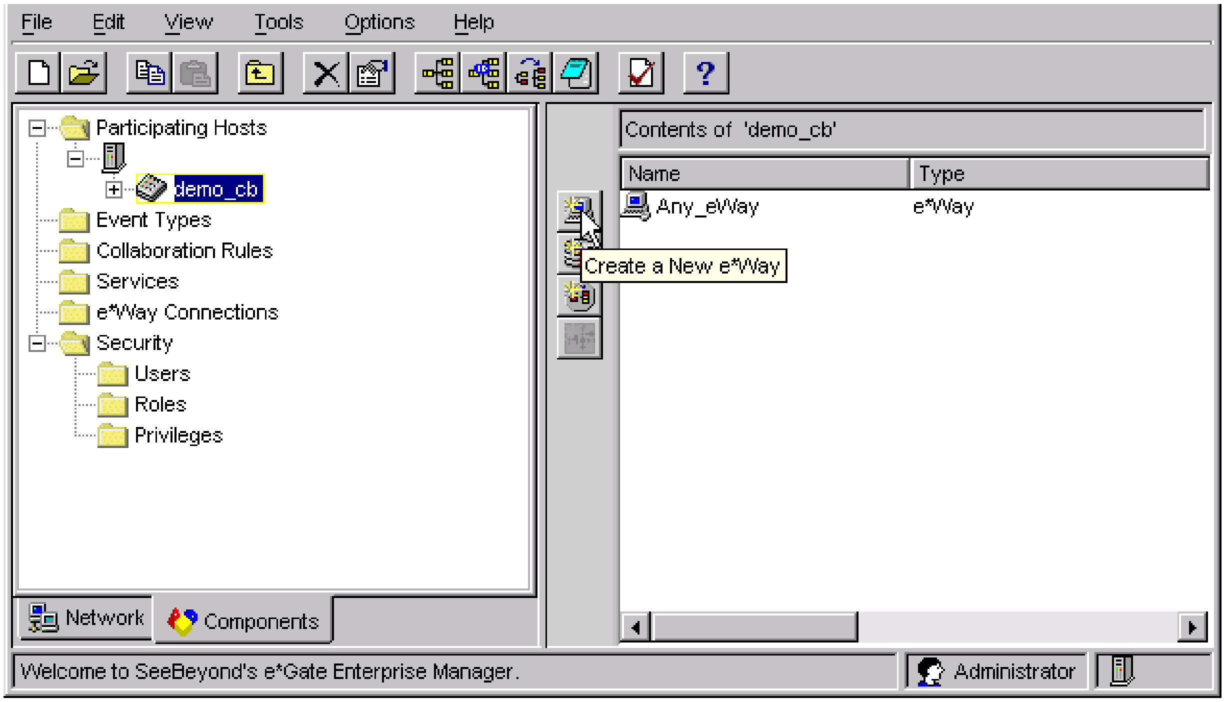
4.2.1 Creating the e*Way

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

To create an e*Way

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

Figure 17 e*Gate Schema Designer Window (Components View)



Contents of 'demo_cb'

Name	Type
Any_eWay	e*Way

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 Schema Designer Navigator's **Components** tab.

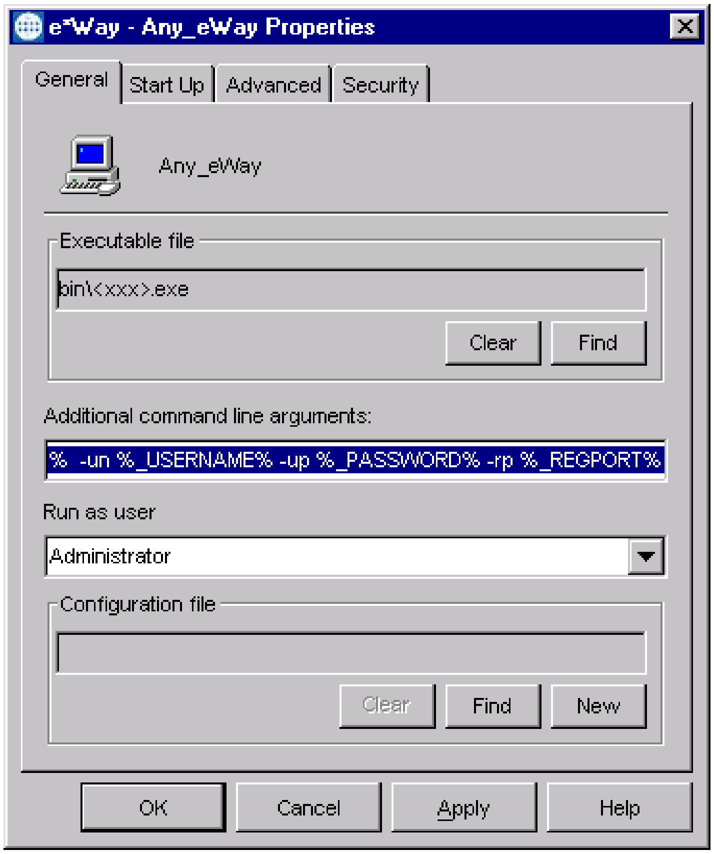
4.2.2 Modifying e*Way Properties

To modify any e*Way properties

1 Right-click on the desired e*Way and select **Properties** to edit the e*Way's properties. The properties dialog opens to the **General** tab (shown in Figure 18).

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

Figure 18 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

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

4.2.3 Configuring the e*Way

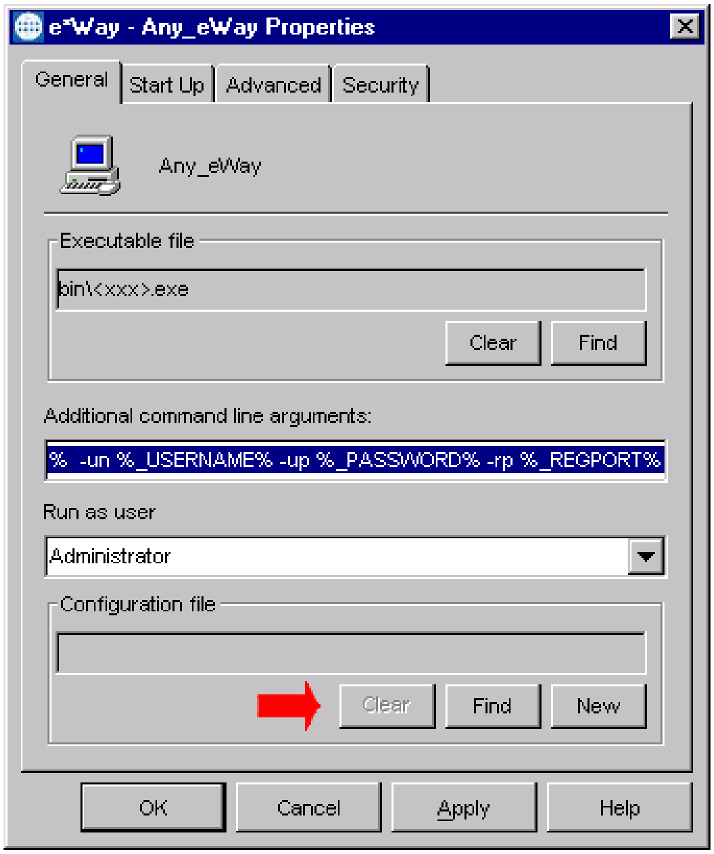
The e*Way's default configuration parameters are stored in an ASCII text file with a **.def** extension. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (**.cfg**) file.

To change e*Way configuration parameters

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

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

Figure 19 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:

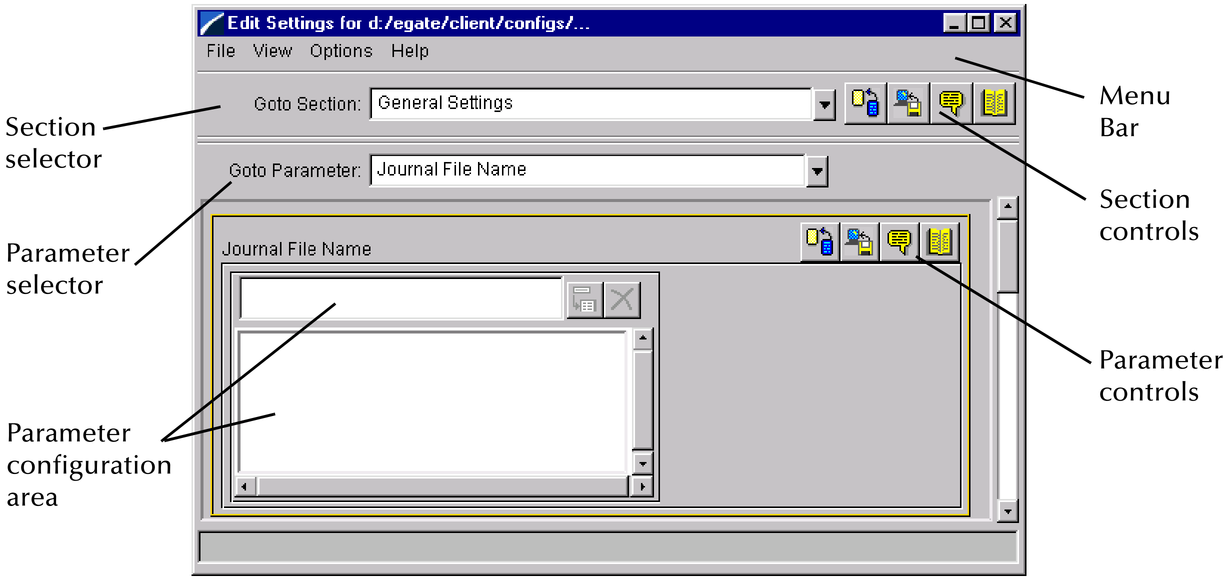
2 Under **Configuration File**, click **New** to create a new file or **Find** to select an existing configuration file. If you select an existing file, an **Edit** button appears. Click this button to edit the currently selected file.

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

Using the e*Way Editor

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

Figure 20 The e*Way Configuration Editor



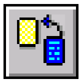
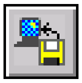
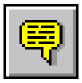
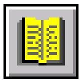
The e*Way Editor controls fall into one of six categories:

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

Section and Parameter Controls

The section and parameter controls are shown in Table 19 below.

Table 19 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

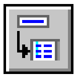
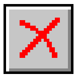
Note: The ***section controls*** *affect* ***all*** *parameters in the selected section, whereas the* ***parameter controls*** *affect only the* ***selected*** *parameter.*

Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Option buttons
- Selection lists, which have controls as described in Table 20

Table 20 Selection List Controls

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

Command-line Configuration

In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

Getting Help

To launch the e*Way Editor's Help system

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

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

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

To display tips regarding the selected Configuration Section

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

To display tips regarding the selected Configuration Parameter

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

Note: *"Tips" are displayed and managed separately from the Help system that launches from the Toolbar's Help menu. You cannot search for Tips within the Help system, or view Help system topics by requesting Tips.*

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

4.2.4 Changing the User Name

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

To change the user name

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

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

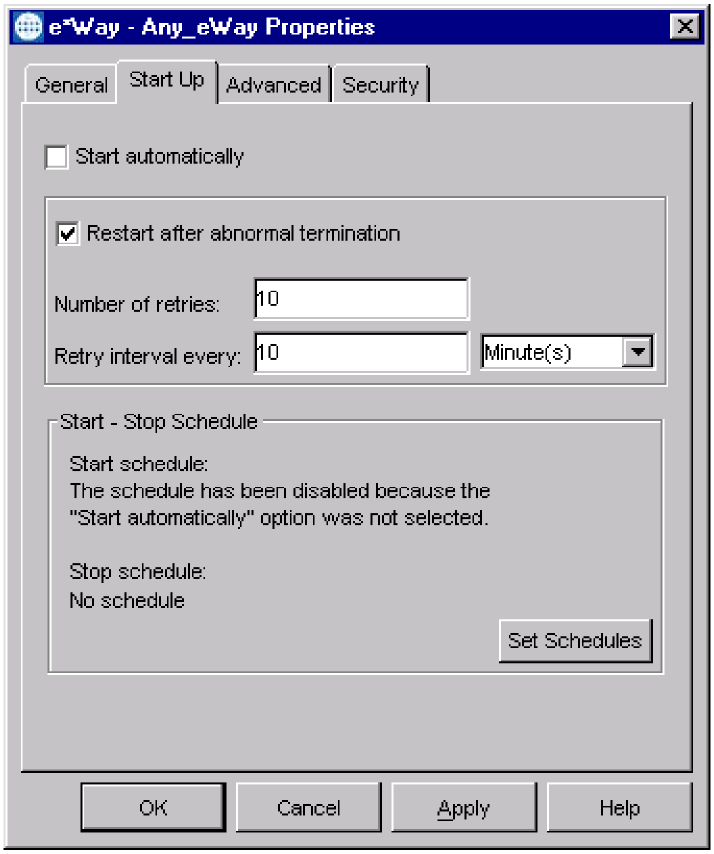
4.2.5 Setting Startup Options or Schedules

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

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

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see Figure 21). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 21 e*Way Properties (Start-Up Tab)



☐ Start automatically

☑ Restart after abnormal termination

Number of retries: 10

Retry interval every: 10 Minute(s)

Start - Stop Schedule

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

Stop schedule:
No schedule

To set the e*Way's startup properties

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

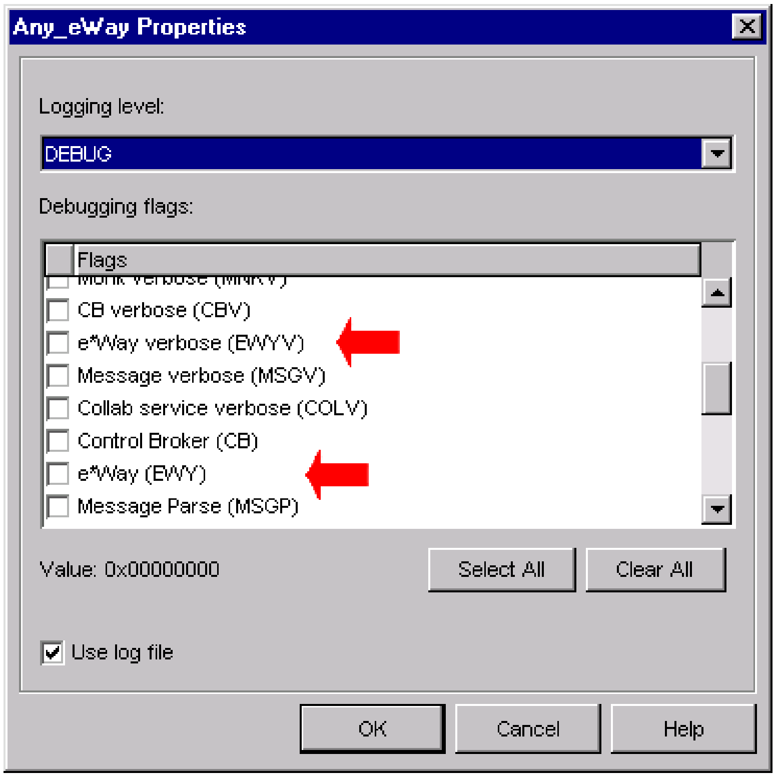
4.2.6 Activating or Modifying Logging Options

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

To set the e*Way debug level and flag

1. Display the e*Way's Properties dialog.
2. Select the **Advanced** tab.
3. Click **Log**. The dialog window appears (see Figure 22).

Figure 22 e*Way Properties (Advanced Tab - Log Option)



Any_eWay Properties

Logging level:

DEBUG

Debugging flags:

Flags

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

Value: 0x00000000

☑ Use log file

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

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

4.2.7 Activating or Modifying Monitoring Thresholds

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

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

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

4.3 Troubleshooting the e*Way

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

4.3.1 Configuration Problems

In the Schema Designer

- Does the e*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration 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 SNA dynamically-loaded libraries. The name of this variable on the different operating systems is:
  - PATH (Windows)
  - LD_LIBRARY_PATH (Solaris)

In the External 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 Schema Manager system to monitor operations and performance.

Chapter 5

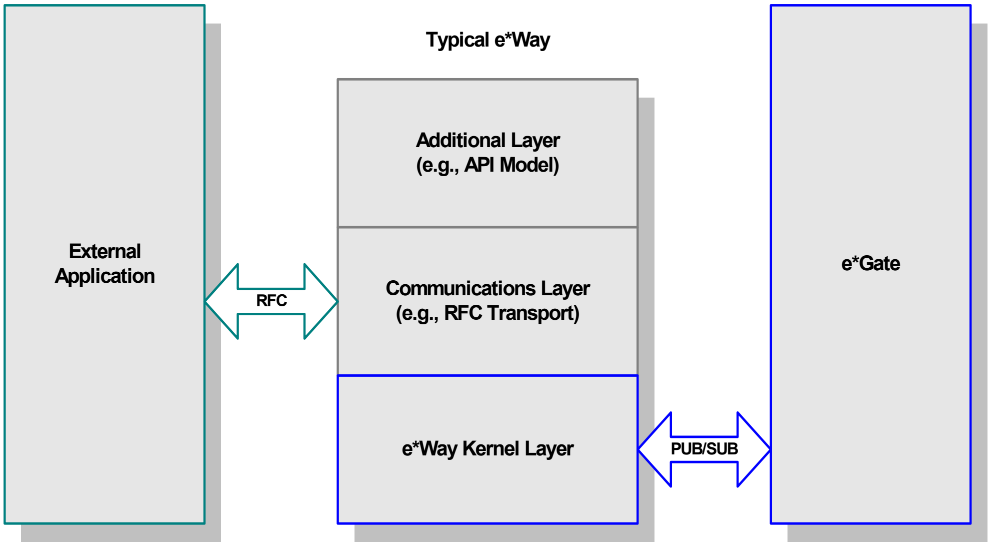
Operational Overview

This chapter contains an overview of the architecture and basic internal processes of the SNA e*Way.

5.1 e*Way Architecture

Conceptually, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers (see Figure 23). 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 23 Typical e*Way Architecture

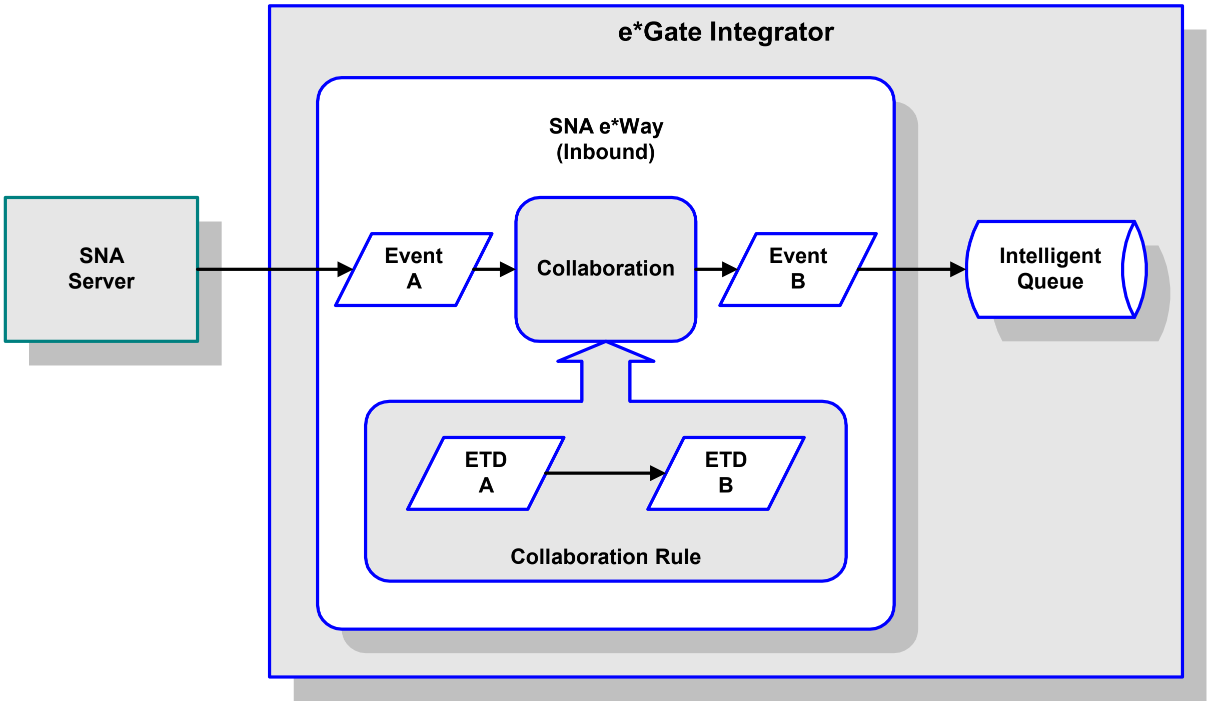

Typical e*Way
External Application
RFC
Additional Layer (e.g., API Model)
Communications Layer (e.g., RFC Transport)
e*Way Kernel Layer
PUB/SUB
e*Gate

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

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

Collaborations execute the business logic that enable the e*Way to do its intended work. In turn, each Collaboration executes a Collaboration Rule, containing the actual instructions to execute the business logic. Each Collaboration that publishes its processed Events internally (within e*Gate Integrator) requires one or more IQs to receive the Events, as shown in Figure 24. Any Collaboration that publishes its processed Events only to an external system does *not* require *any* IQs.

Figure 24 Collaborations and IQs



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

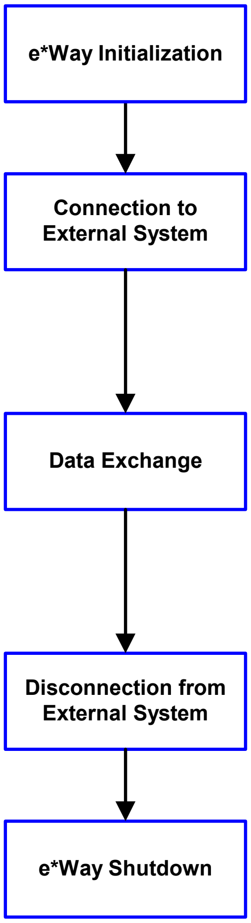
5.2 Basic e*Way Processes

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

The most basic processes carried out by an e*Way are listed in Figure 25. 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 25 Basic e*Way Processes

Process


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

Monk Configuration Sections

Startup Function on page 72 and on page 90 (also see **Monk Environment Initialization File** on page 71 and on page 89).

External Connection Establishment Function on page 74 and on page 92, and **External Connection Verification Function** on page 75 and on page 93.

Event-driven Data Exchange

Process Outgoing Message Function on page 72 and on page 90.

Schedule-driven Data Exchange

Exchange Data with External Function on page 73 and on page 91;
Positive Acknowledgment Function on page 76 and on page 94;
Negative Acknowledgment Function on page 76 and on page 94.

External Connection Shutdown Function on page 75 and on page 93.

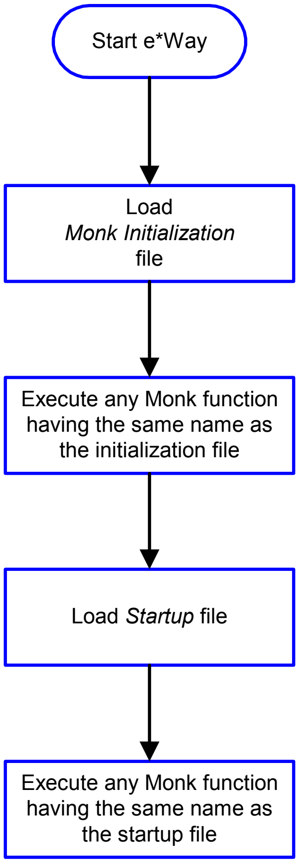
Shutdown Command Notification Function on page 77 and on page 95.

A series of diagrams on the next several pages illustrate the interaction and operation of these functions during the specified processes. Configuring the parameters associated with these functions is covered in **Chapter 6** and **Chapter 7**, while the functions themselves are described in **Chapter 8**.

Initialization Process

Figure 26 illustrates the e*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

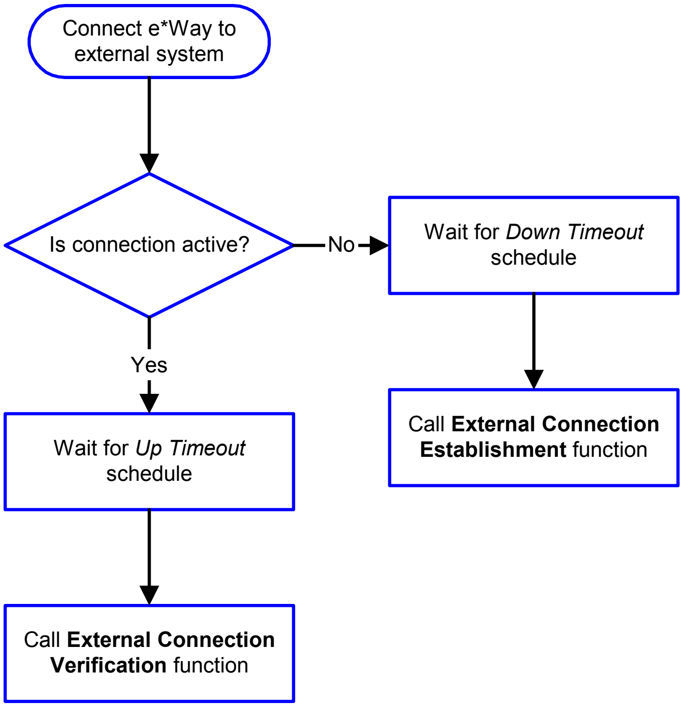
Figure 26 Initialization Process


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

Figure 27 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 29** *on page 61 and* **Figure 28** *on page 60 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 130 and* **send-external-down** *on page 130 for more information.*

Data Exchange Process

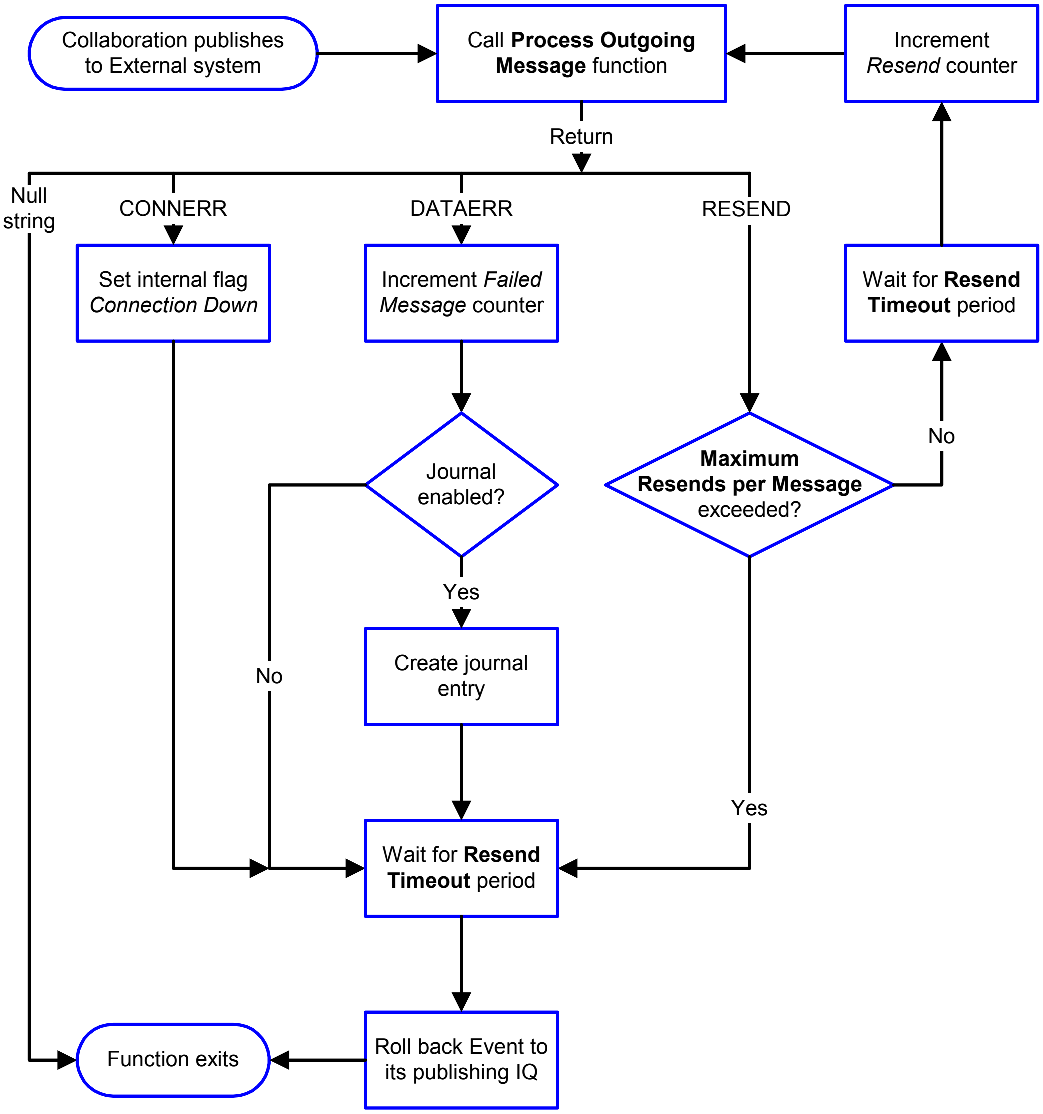
Event-driven

Figure 28 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.

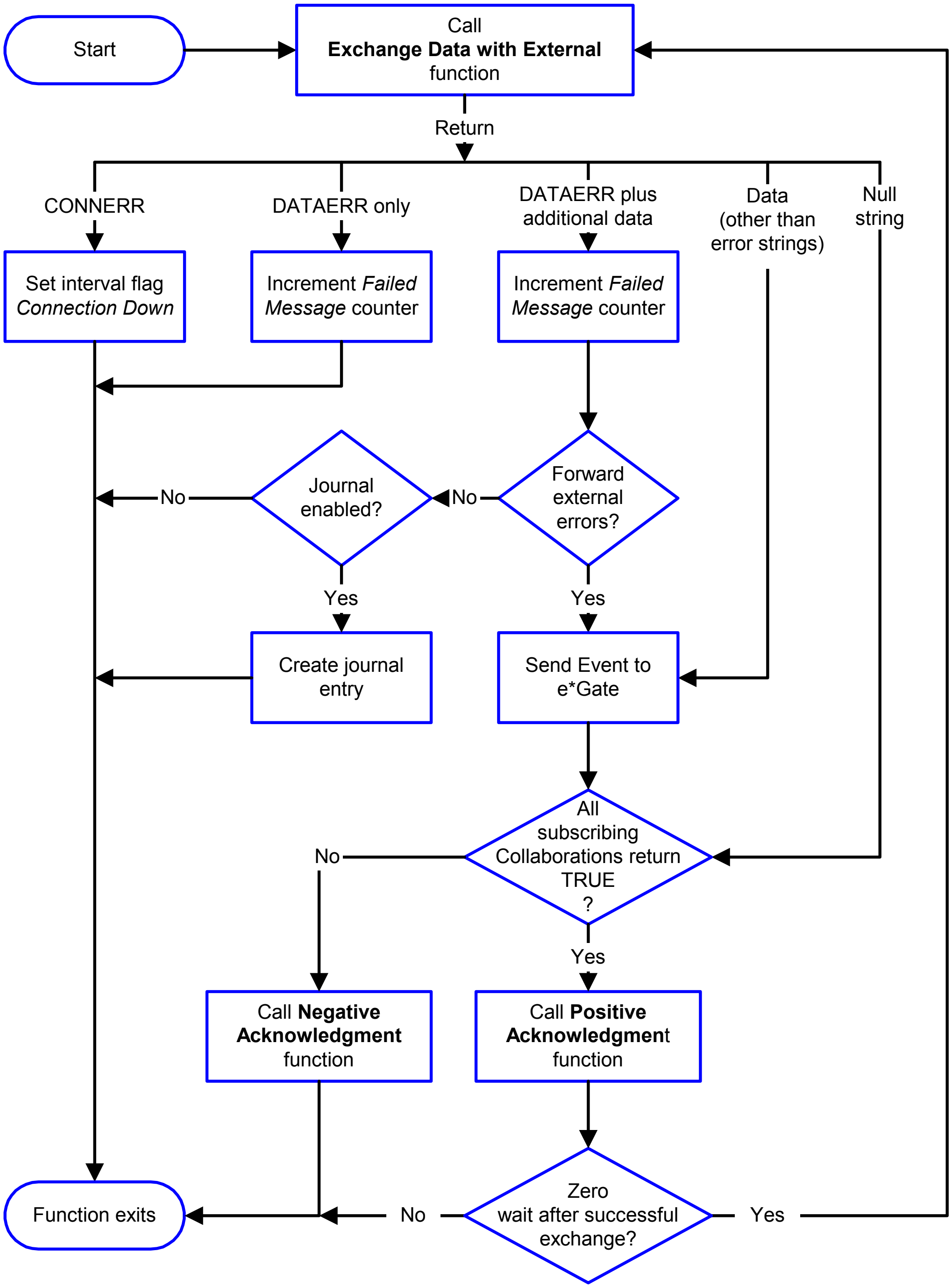
Figure 28 Event-Driven Data Exchange Process


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

Schedule-driven

Figure 29 illustrates how the e*Way's schedule-driven data exchange process works for incoming data, using the **Exchange Data with External Function**, **Positive Acknowledgment Function**, and **Negative Acknowledgment Function**.

Figure 29 Schedule-Driven Data Exchange Process



Start can occur in any of the following ways:

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

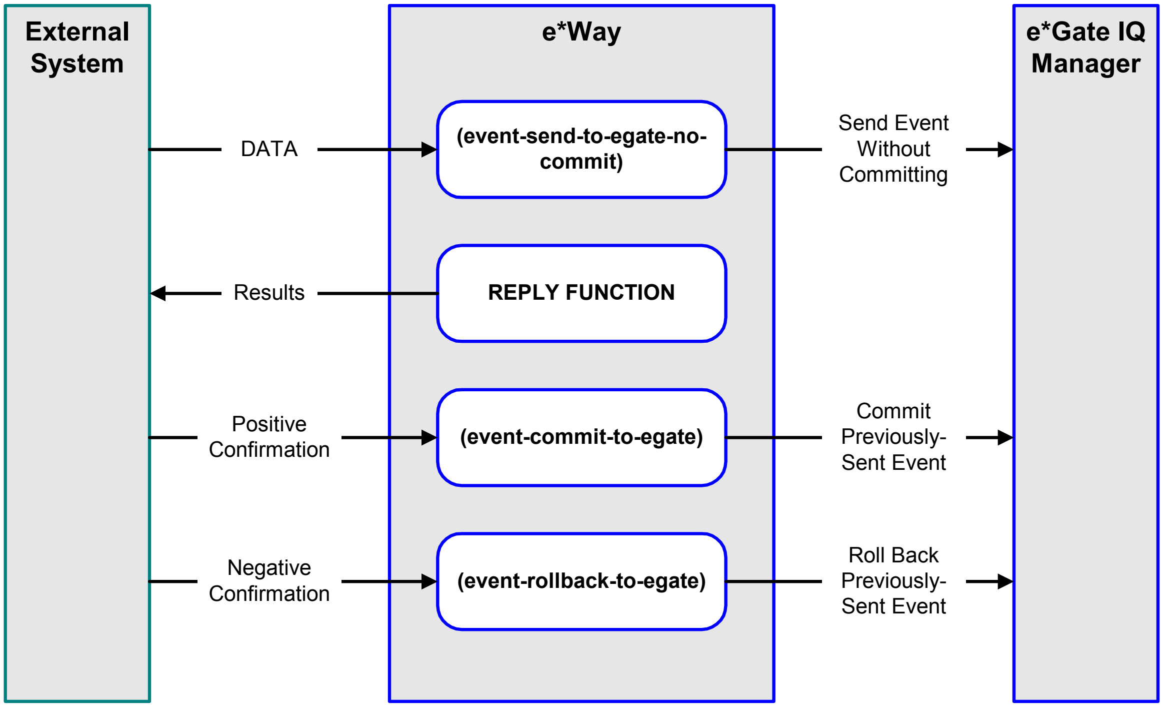
*Send Events to e*Gate* can be implemented using any of the following Monk functions:

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

The last of these is used when confirmation of correct transmission is required from the external system. In this case, the e*Way sends information back to the external system after receiving data. Depending upon whether the acknowledgment is positive or negative, you subsequently use one of the following functions to complete the process (see Figure 30):

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

Figure 30 Send Event to e*Gate with Confirmation

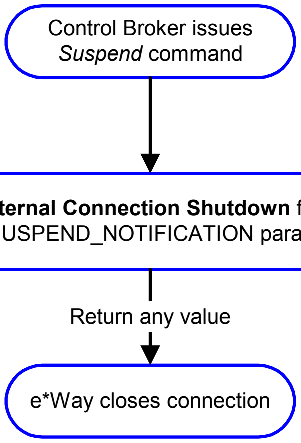

External System
e*Way
e*Gate IQ Manager
DATA
(event-send-to-egate-no-commit)
Send Event Without Committing
Results
REPLY FUNCTION
Positive Confirmation
(event-commit-to-egate)
Commit Previously-Sent Event
Negative Confirmation
(event-rollback-to-egate)
Roll Back Previously-Sent Event

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

Disconnect from External Process

Figure 31 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

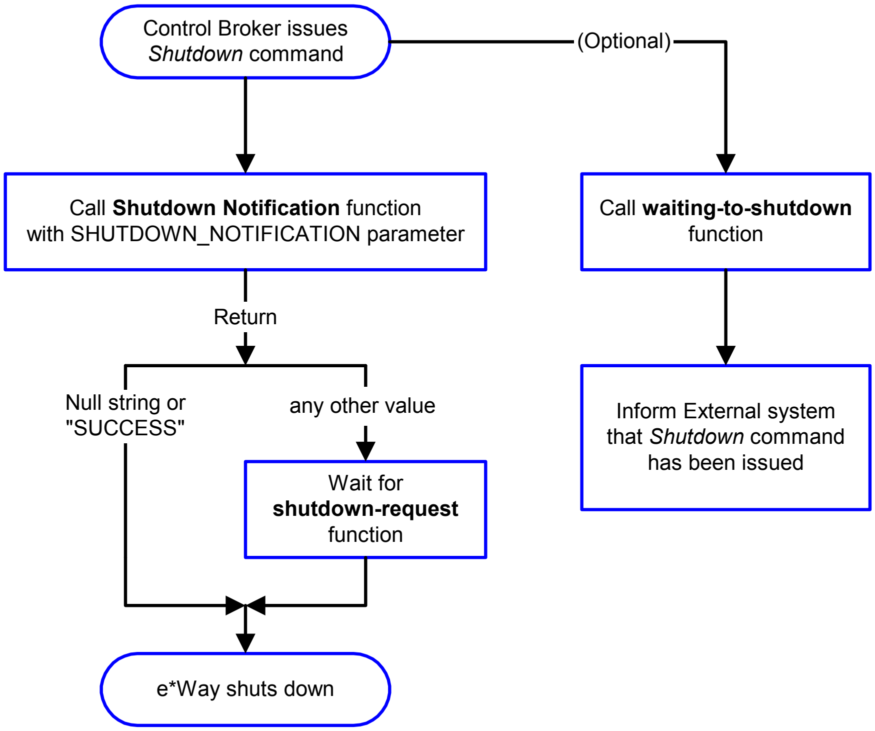
Figure 31 Disconnect Process


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

Shutdown Process

Figure 32 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

Figure 32 Shutdown Process


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

Chapter 6

Configuration Parameters (LU6.2)

This chapter describes the LU6.2 configuration parameters for the e*Way Intelligent Adapter for e*Gate.

6.1 Overview

The e*Way's configuration parameters are set using the e*Way Editor; see **Configuring the e*Way** on page 45 for procedural information. The e*Gate e*Way's configuration parameters are organized into the following sections. The default configurations are provided in **stcewsna.def**.

General Settings on page 65

Communication Setup on page 67

Monk Configuration on page 70

SNA Client Configuration on page 79

6.2 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

A valid filename, optionally including an absolute path (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is stored in the e*Gate **SystemData** directory. There is no default value for this parameter.

Additional Information

An Event is journaled for the following conditions:

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

See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

Max Resends Per Message

Description

Specifies the number of times the e*Way attempts to resend a message (Event) to the external system after receiving an error. When this maximum is reached, the e*Way waits for the number of seconds specified by the **Resend Timeout** parameter, and then rolls back the Event to its publishing IQ.

Required Values

An integer from **1** to **1,024** (omit the comma). The default value is **5**.

Max Failed Messages

Description

Specifies the maximum number of failed Events that the e*Way allows. When the specified number of failed Events is reached, the e*Way shuts down and exits.

Required Values

An integer from **1** to **1,024** (omit the comma). The default value is **3**.

Forward External Errors

Description

Description

Selects whether or not error messages received from the external system that begin with the string **"DATAERR"** are queued to the e*Way's configured queue. See **Exchange Data with External Function** on page 73 for more information.

Required Values

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

6.3 Communication Setup

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

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

Exchange Data Interval

Description

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

Required Values

An integer from **0** to **86,400** (omit the comma). The default value is **120**.

Additional Information

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

See also

Start Exchange Data Schedule on page 68

Stop Exchange Data Schedule on page 69

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 value is **No**.

Additional Information

- If this parameter is set to **Yes**, and the previous exchange function returned data, the e*Way invokes the **Exchange Data with External Function** immediately
- If it is desired to invoke the **Exchange Data with External Function** again as soon as possible when data is **not** queued to e*Gate via the return mechanism, the e*Way Kernel Monk function **insert-exchange-data-event** can be called directly (prior to leaving the exchange function) to accomplish this
- If this parameter is set to **No**, the e*Way always waits the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External Function**

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

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

Other Requirements

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

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

Additional Information

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

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

If *no* ACK/NAK is pending and the connection *is* active, the e*Way immediately executes the **Exchange Data with External Function**. Thereafter, the **Exchange Data with External Function** is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

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

Down Timeout

Description

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

Required Values

An integer from **1** to **86,400** (omit the comma). The default value 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 from **1** to **86,400** (omit the comma). The default is **15**.

Resend Timeout

Description

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

Required Values

An integer from **1** to **86,400** (omit the comma). The default is **10**.

6.4 Monk Configuration

The parameters in this section help you set up the information required by the e*Way to utilize Monk for communication with the external system. The *functions* that you specify within this section are Monk functions that the e*Way calls automatically as part of its normal operations. The functions are not called under user control.

All the configuration options in this section—the functions or variables defined, and the additional path information—are loaded into a separate Monk environment than is used by the e*Way's Collaborations and its Collaboration Rules scripts. You cannot access any of these functions, variables, or path information from Collaboration Rules scripts.

Specifying Function or File Names

For those parameters that accept a file or the name of a Monk function, the e*Way presumes that the name of the file is the same as the name of the function to be executed, plus a **.monk** extension. For example, the file **startup.monk** should contain the definition for the function **startup**. If path information is specified, that path is appended to the **Load Path**.

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

- .monk
- .tsc
- .dsc

Specifying Multiple Directories

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

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

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

Load Path

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

Additional Path

Description

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

Required Values

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

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

Additional information

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

Auxiliary Library Directories

Description

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

Required Values

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

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 **sna-init**.

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

Returns

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

Additional information

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

Startup Function

Description

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

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **sna-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 **sna-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 Schema Designer).
- Once this function has been called with a *non-null string*, the e*Way does not process another Event until the current Event has been completely processed.

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

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is invoked automatically by the **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 **sna-incoming**.

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

Returns

- A *null string* (“”) indicates that the data exchange was completed successfully, but with no resultant data sent back to the e e*Gate system
- A string beginning with **CONNERR** indicates that there is a problem with the connection to the external system
- A string beginning with **DATAERR** indicates that there is a problem with the message (Event) data itself. If the error string contains data beyond the keyword, the entire string is queued to e*Gate if an inbound Collaboration is so configured and **Forward External Errors** is set to **Yes**. Queueing, however, is performed without the subsequent sending of a **ACK** or **NAK** to the external system.
- Any other string indicates that the contents of the string are packaged as an inbound Event

Additional Information

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

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

External Connection Establishment Function

Description

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

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **sna-conn-establish**.

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

Returns

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

External Connection Verification Function

Description

Specifies a Monk function that the e*Way calls when its internal variables show that the connection to the external system is up. It is executed according to the interval specified within the **Up Timeout** parameter, and is called *only* according to this schedule.

Required Values

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

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

Returns

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

Additional Information

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

External Connection Shutdown Function

Description

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **sna-conn-shutdown**.

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

Input

A string indicating the purpose for shutting down the connection.

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

Returns

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

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

Positive Acknowledgment Function

Description

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **sna-pos-ack**.

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

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 **sna-neg-ack**.

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

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 **sna-conn-shutdown**.

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

Input

When the Control Broker issues a shutdown command to the e*Way, the e*Way calls this function with the string **"SHUTDOWN_NOTIFICATION"** passed as a parameter.

Returns

- A *null string* or **"SUCCESS"** indicates that the shutdown can occur immediately
- Any other string indicates that shutdown must be postponed; once postponed, shutdown does not proceed until the Monk function **shutdown-request** is executed

Additional Information

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

6.5 SNA Client Configuration

The parameters in this section provide the information required by the Generic Monk e*Way to support SNA LU6.2.

SYMDESTNAME

Description

Specifies the symbolic destination name on which the SNA client is running.

Required Values

A string; this field is *case sensitive* and can contain up to **64** ASCII characters.

Note: *This parameter is required; you must* ***not*** *leave this field blank.*

LOCALTPNAME

Description

Specifies the name of the local TP that is running on the local LU.

Required Values

A string; this field is *case sensitive* and should be **8** characters in length.

Note: *This parameter is required; you must* ***not*** *leave this field blank.*

LOCALLUNAME

Description

Specifies the name of the local LU as defined for the SunLink 6.2 server.

Required Values

A string; this field is *case sensitive*.

Note: *This parameter is required for Sunlink P2P LU6.2 version 9.1, and is ignored on other platforms.*

PacketSize

Description

Specifies the number of bytes per packet of data.

Required Values

An integer from **0** to **864,000** (omit the comma). The default value is **1024**.

Timeout

Description

Specifies the number of milli-seconds to wait for a response, when making requests to the server.

Required Values

An integer from **0** to **864,000** (omit the comma). The default value is **50000**.

Use Ack Nak

Description

Specifies whether or not to use **ACK** and **NAK** for **Request Reply**.

Required Values

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

Ack String

Description

Specifies the Positive acknowledgment value.

Required Values

A string. The default value is **ACK**.

Nak String

Description

Specifies the Negative acknowledgment value.

Required Values

A string. The default value is **NAK**.

Request Reply

Description

Specifies whether or not the Process Outgoing Function waits for a reply and posts that reply to e*Gate.

Required Values

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

Initialize Conversation

Description

Specifies whether to initialize a conversation with the remote LU, or to accept conversation from the remote LU.

Required Values

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

- Set the value to **Yes** to initialize a conversation with the remote LU.
- Set the value to **No** to accept a conversation from a remote LU.

Data Flow

Description

Specifies the direction of data flow.

Required Values

Inbound or **Outbound**. The default value is **Outbound**.

- Set the value to **Outbound** to allow the local LU to send data to the partner LU.
- Set the value to **Inbound** to allow the local LU to receive data from the partner LU.

Synchronization Level

Description

Specifies the synchronization level of the conversation.

Required Values

Confirm or **None**. The default value is **Confirm**.

- Select **Confirm** to set the synchronization level parameter, **CM_SYNC_LEVEL**, to **CM_CONFIRM**.
- Select **None** to set the synchronization level parameter, **CM_SYNC_LEVEL**, to **CM_NONE**.

Chapter 7

Configuration Parameters (LUA)

This chapter describes the LUA configuration parameters for the e*Way Intelligent Adapter for e*Gate.

7.1 Overview

The e*Way's configuration parameters are set using the e*Way Editor; see **Configuring the e*Way** on page 45 for procedural information. The e*Gate e*Way's configuration parameters are organized into the following sections. The default configurations are provided in **stcewsnalu0.def**.

General Settings on page 83

Communication Setup on page 85

Monk Configuration on page 88

SNA LUA Client Configuration on page 97

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 (for example, **c:\temp\filename.txt**). If an absolute path is not specified, the file is stored in the e*Gate **SystemData** directory. There is no default value for this parameter.

Additional Information

An Event is Journaled for the following conditions:

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

See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

Max Resends Per Message

Description

Specifies the number of times the e*Way attempts to resend a message (Event) to the external system after receiving an error. When this maximum is reached, the e*Way waits for the number of seconds specified by the **Resend Timeout** parameter, and then rolls back the Event to its publishing IQ.

Required Values

An integer from **1** to **1,024** (omit the comma). The default value is **5**.

Max Failed Messages

Description

Specifies the maximum number of failed Events that the e*Way allows. When the specified number of failed Events is reached, the e*Way shuts down and exits.

Required Values

An integer from **1** to **1,024** (omit the comma). The default value is **3**.

Forward External Errors

Description

Description

Selects whether or not error messages received from the external system that begin with the string **"DATAERR"** are queued to the e*Way's configured queue. See **Exchange Data with External Function** on page 91 for more information.

Required Values

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

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 Schema Designer controls when the e*Way executable runs. The schedule you set within the parameters discussed in this section (using the e*Way Editor) determines when data are exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Exchange Data Interval

Description

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

Required Values

An integer from **0** to **86,400** (omit the comma). The default value is **120**.

Additional Information

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

See also

Down Timeout on page 87

Stop Exchange Data Schedule on page 86

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 value is **No**.

Additional Information

- If this parameter is set to **Yes**, and the previous exchange function returned data, the e*Way invokes the **Exchange Data with External Function** immediately
- If it is desired to invoke the **Exchange Data with External Function** again as soon as possible when data is **not** queued to e*Gate via the return mechanism, the e*Way Kernel Monk function **insert-exchange-data-event** can be called directly (prior to leaving the exchange function) to accomplish this
- If this parameter is set to **No**, the e*Way always waits the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External Function**

Start Exchange Data Schedule

Description

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

Required Values

One or more schedules. The schedule can specify a date, time, or frequency (such as yearly, weekly, monthly, daily, or every ***n*** seconds). There is no default value for this parameter.

Also required

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

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

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

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One or more schedules. The schedule can specify a date, time, or frequency (such as yearly, weekly, monthly, daily, or every ***n*** seconds).

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 from **1** to **86,400** (omit the comma). The default value 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 from **1** to **86,400** (omit the comma). The default value is **15**.

Resend Timeout

Description

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

Required Values

An integer from **1** to **86,400** (omit the comma). The default value is **10**.

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. The *functions* that you specify within this section are Monk functions that the e*Way calls automatically as part of its normal operations. The functions are not called under user control.

All the configuration options in this section—the functions or variables defined, and the additional path information—are loaded into a separate Monk environment than is used by the e*Way's Collaborations and its Collaboration Rules scripts. You cannot access any of these functions, variables, or path information from Collaboration Rules scripts.

Specifying Function or File Names

For those parameters that accept a file or the name of a Monk function, the e*Way presumes that the name of the file is the same as the name of the function to be executed, plus a **.monk** extension. For example, the file **startup.monk** should contain the definition for the function **startup**. If path information is specified, that path is appended to the **Load Path**.

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

- .monk
- .tsc
- .dsc

Specifying Multiple Directories

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

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

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

Load Path

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

Additional Path

Description

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

Required Values

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

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

Additional information

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

Auxiliary Library Directories

Description

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

Required Values

A pathname, or a series of paths separated by semicolons. The default value is **monk_library/ewsnalu0**.

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 **snalu0-init**.

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

Returns

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

Additional information

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

Startup Function

Description

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

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **snalu0-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 **snalu0-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 Schema Designer).
- Once this function has been called with a *non-null string*, the e*Way does not process another Event until the current Event has been completely processed.

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

Exchange Data with External Function

Description

Specifies a Monk function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is invoked automatically by the **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 **snalu0-incoming**.

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

Returns

- A *null string* ("") indicates that the data exchange was completed successfully, but with no resultant data sent back to the e e*Gate system
- A string beginning with **CONNERR** indicates that there is a problem with the connection to the external system
- A string beginning with **DATAERR** indicates that there is a problem with the message (Event) data itself. If the error string contains data beyond the keyword, the entire string is queued to e*Gate if an inbound Collaboration is so configured and **Forward External Errors** is set to **Yes**. Queueing, however, is performed without the subsequent sending of a **ACK** or **NAK** to the external system.
- Any other string indicates that the contents of the string are packaged as an inbound Event

Additional Information

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

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

External Connection Establishment Function

Description

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

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **snalu0-conn-establish**.

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

Returns

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

External Connection Verification Function

Description

Specifies a Monk function that the e*Way calls when its internal variables show that the connection to the external system is up. It is executed according to the interval specified within the **Up Timeout** parameter, and is called *only* according to this schedule.

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **snalu0-conn-verify**.

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

Returns

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

Additional Information

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

External Connection Shutdown Function

Description

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **snalu0-shutdown**.

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

Input

A string indicating the purpose for shutting down the connection.

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

Returns

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

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

Positive Acknowledgment Function

Description

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

Required Values

The name of a Monk function or the name of a file containing a Monk function. The default value is **snalu0-pos-ack**.

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

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 **snalu0-neg-ack**.

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

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 **snalu0-conn-shutdown**.

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

Input

When the Control Broker issues a shutdown command to the e*Way, the e*Way calls this function with the string **"SHUTDOWN_NOTIFICATION"** passed as a parameter.

Returns

- A *null string* or **"SUCCESS"** indicates that the shutdown can occur immediately
- Any other string indicates that shutdown must be postponed; once postponed, shutdown does not proceed until the Monk function **shutdown-request** is executed

Additional Information

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

7.5 SNA LUA Client Configuration

The parameters in this section provide the information required by the Generic Monk e*Way to support SNA LUA (including LU0).

Local LU Name

Description

Specifies the Local LU defined on VTAM for local host.

Required Values

A string; this field is *case sensitive*. There is no default value for this parameter.

Note: *This parameter is required; you must* ***not*** *leave this field blank.*

Max Message Size

Description

Specifies the maximum number of bytes per packet of data. This number also determines the size of the buffers.

Required Values

An integer from **1** to **864,000** (omit the comma). The default value is **1024**

Note: *This parameter is required; you must* ***not*** *leave this field blank.*

Receive Timeout

Description

Specifies the number of milli-seconds to wait when reading from the SNA server.

Required Values

An integer from **1** to **864,000** (omit the comma). The default is **50000**.

Control Bytes

Description

Specifies the number bytes to preserve at the beginning of the data. These are generally used for information such as the MUX header.

Required Values

An integer from **0** to **864,000** (omit the comma). The default is **0**.

Chapter 8

API Functions

This chapter describes the various API functions used by the e*Gate e*Way.

8.1 Overview

The e*Gate e*Way's functions fall into the following categories:

- **Native e*Way Functions**
  - **LU6.2** on page 99
  - **LUA** on page 108
- **Standard e*Way Functions**
  - **LU6.2** on page 113
  - **LUA** on page 120
- **Generic e*Way Functions** on page 126

8.2 Native e*Way Functions

The functions described in this section control the SNA e*Way's interaction with SNA, and can only be called from within a Collaboration Rules script.

8.2.1 LU6.2

The SNA e*Way's native Monk functions for LU6.2 are:

sna-accept-conversation on page 99

sna-change-state on page 100

sna-change-state-no-synch on page 100

sna-confirmed on page 101

sna-client-connect on page 102

sna-client-connect-no-synch on page 102

sna-client-disconnect on page 103

sna-client-isconnected on page 103

sna-client-recv on page 104

sna-client-recv-no-synch on page 105

sna-client-send on page 105

sna-client-send-no-synch on page 106

sna-accept-conversation

Description

Allows the client to accept conversation by means of the following sequence.

1. Calls CMSLTP to specify the local TP name.
2. Calls CMACCP to accept the conversation.
3. Calls CMWAIT to wait for the local LU to attach the conversation.

Signature

```
(sna-accept-conversation LocalTPName)
```

Parameters

Name	Type	Description
LocalTPName	string	The Local TP Name associated with the SNA Server.

Returns

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

Throws

None.

Location

```
stc_monksna.dll
```

sna-change-state

Description

Changes the state of the SNA conversation as follows:

- If the parameter *State* = SEND, calls CMPTR to change the state to RECEIVE
- If the parameter *State* = RECEIVE, calls CMRTS to change the state to SEND
  - After calling CMRTS, calls CMCFMED to get confirmation that the request to send was received

Signature

```
(sna-change-state ServerHandle State)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server.
State	string	The Server State (send or receive).

Returns

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

Throws

None.

Additional Information

- Before a send or receive can be called, the conversation must be in the correct state. For the client to send an Event to the server, the state must be **send**. In order to receive an Event from the server, the state must be **receive**. This *must* be synchronized with the server. Neither a **send** nor a **receive** occurs unless both TPs are synchronized.
- If the conversation is already in the state being requested, an error is returned.

Location

```
stc_monksna.dll
```

sna-change-state-no-synch

Description

Changes the state of the SNA conversation with no synchronization calls.

Signature

```
(sna-change-state ServerHandle State)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server.
State	string	The Server State (send or receive).

Returns

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

Throws

None.

Additional Information

- Before a send or receive can be called, the conversation must be in the correct state. For the client to send an Event to the server, the state must be **send**. In order to receive an Event from the server, the state must be **receive**. This *must* be synchronized with the server. Neither a **send** nor a **receive** occurs unless both TPs are synchronized.
- If the conversation is already in the state being requested, an error is returned.

sna-confirmed

Description

Calls CMCFMD to reply to a confirmation request from the partner program to verify that there was no error detected by the local program within the data received.

Signature

```
(sna-confirmed ServerHandle)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server

Returns

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

Throws

None.

Additional Information

SNA requires that the local and the partner program issue a 'confirmed' call after each instance of data received. The conversation is blocked until such confirmation is received. The 'confirmed' call synchronizes the processing of the two TPs.

Location

```
stc_monksna.dll
```

sna-client-connect

Description

Opens a connection to the specified server by means of the following sequence.

1. Calls CMINIT, to initialize the conversation with the partner LU.
2. Calls CMSPM, to set the processing mode to CM_BLOCKING.
3. Calls CMSSL, to set the synchronization level to CM_CONFIRM.
4. Calls CMALLC, to allocate the conversation with the partner LU.

Signature

```
(sna-client-connect SymDestName)
```

Parameters

Name	Type	Description
SymDestName	string	The Symbolic Destination Name associated with the SNA Server (see **SYMDESTNAME** on page 79).

Returns

Returns the handle to the SNA Server.

Throws

None.

Location

```
stc_monksna.dll
```

sna-client-connect-no-synch

Description

Opens a connection to the specified server and defaults the synchronization level to CM_NONE.

Signature

```
(sna-client-connect SymDestName)
```

Parameters

Name	Type	Description
SymDestName	string	The Symbolic Destination Name associated with the SNA Server (see **SYMDESTNAME** on page 79).

Returns

Returns the handle to the SNA Server.

Throws

None.

Location

```
stc_monksna.dll
```

sna-client-disconnect

Description

Closes the connection to the SNA server.

Signature

```
(sna-client-disconnect ServerHandle)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server.

Returns

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

Throws

None.

Additional Information

Generally, the TP that is sending data should deallocate the conversation; however, if you are receiving data and want to disconnect, first call **sna-change-state “SEND”**.

Location

```
stc_monksna.dll
```

sna-client-isconnected

Description

Verifies that the connection to the SNA server is open.

Signature

```
(sna-client-isconnected ServerHandle)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server

Returns

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

Throws

None.

Location

```
stc_monksna.dll
```

sna-client-recv

Description

Contacts the specified SNA server to advise that it is ready to receive any data (Event) that is available from the server.

1. Calls CMRCV to receive data from partner LU.
2. Calls CMCFMD to send confirmation to the partner that the data was received successfully.

Signature

```
(sna-client-recv ServerHandle PacketSize Timeout)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server.
PacketSize	integer	The size of the packet in bytes (see **PacketSize** on page 79)
Timeout	integer	The amount of milli-seconds to wait for a response from the Server before a Timeout is issued (see **Timeout** on page 80).

Returns

Returns a string representing the Event.

Throws

None.

Additional Information

The states must be synchronized prior to making this call. The local program state must be in the 'receive' mode.

Location

```
stc_monksna.dll
```

sna-client-recv-no-synch

Description

Contacts the specified SNA server to advise that it is ready to receive any data (Event) that is available from the server. There are no synchronization calls with this function.

Signature

```
(sna-client-recv ServerHandle PacketSize Timeout)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server.
PacketSize	integer	The size of the packet in bytes (see **PacketSize** on page 79).
Timeout	integer	The amount of milli-seconds to wait for a response from the Server before a Timeout is issued (see **Timeout** on page 80).

Returns

Returns a string representing the Event.

Throws

None.

Additional Information

The states must be synchronized prior to making this call. The local program state must be in the 'receive' mode.

Location

```
stc_monksna.dll
```

sna-client-send

Description

Sends an Event to the specified SNA server.

Signature

```
(sna-client-send ServerHandle Event)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server
Event	string	The Event to be sent.

Returns

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

Throws

None.

Additional Information

The states must be synchronized prior to making this call. The local program state must be in the 'send' mode.

Location

```
stc_monksna.dll
```

sna-client-send-no-synch

Description

Sends an Event to the specified SNA server.

1. Calls CMSEND to send data to the partner LU, and then
2. Issues CMCFM to request a confirmation from the partner LU that the data sent was received successfully.

Signature

```
(sna-client-send ServerHandle Event)
```

Parameters

Name	Type	Description
ServerHandle	opaque handle	The handle to the SNA Server
Event	string	The Event to be sent.

Returns

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

Throws

None.

Additional Information

No synchronization is required prior to making this call. The local program state must be in the 'send' mode.

Location

```
stc_monksna.dll
```

8.2.2 LUA

The SNA e*Way's native Monk functions for LUA (and LU0) are:

snalu0-connect on page 108

snalu0-disconnect on page 108

snalu0-isconnected on page 109

snalu0-send on page 110

snalu0-recv on page 110

snalu0-get-property on page 111

snalu0-set-property on page 111

snalu0-connect

Description

Calls RUI_INIT, which notifies VTAM that a connection is desired.

Signature

```
(snalu0-connect luName timeout)
```

Parameters

Name	Type	Description
luName	string	A zero-delimited string specifying the local LU name (see **Local LU Name** on page 97).
timeout	int	Timeout in milliseconds to wait for a response from the server before timing out (see **Receive Timeout** on page 97).

Returns

Returns a handle for subsequent SNA calls.

Throws

None.

Examples

```
(define hSNA (snalu0-connect "T1860C01")
```

Location

```
monksnalua.monk
```

snalu0-disconnect

Description

Calls RUI_INIT, which notifies VTAM that the connection no longer desired.

Signature

```
(snalu0-disconnect snaHandle)
```

Parameters

Name	Type	Description
snaHandle	Opaque handle	The handle to the SNA Server.

Returns

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

Throws

None.

Examples

```
(snalu0-disconnect hSNA)
```

Location

```
monksnalua.monk
```

snalu0-isconnected

Description

Sends a status to SNA Server to verify that the connection handle is still valid.

Signature

```
(snalu0-isconnected snaHandle)
```

Parameters

Name	Type	Description
snaHandle	Opaque handle	The handle to the SNA Server.

Returns

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

Throws

None.

Examples

```
(if (snalu0-isconnected hSNA)
    (display “handle still good\n”)
    (display “handle bad\n”)
)
```

Location

```
monksnalua.monk
```

snalu0-send

Description

Sends an Event to the specified SNA Server.

Signature

```
(snalu0-send snaHandle event)
```

Parameters

Name	Type	Description
snaHandle	Opaque handle	The handle to the SNA Server.
event	String	Data to send to the SNA Server.

Returns

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

Throws

None.

Examples

```
(snalu0-send hSNA "Hello There")
```

Location

```
monksnalua.monk
```

snalu0-recv

Description

Receives an Event from the specified SNA Server.

Signature

```
(snalu0-recv snaHandle packetSize timeout)
```

Parameters

Name	Type	Description
snaHandle	Opaque handle	The handle to the SNA Server.
packetSize	Integer	The size of the packet in bytes to read.
timeout	Integer	Timeout in milliseconds to wait for a response from the server before timing out (see **Receive Timeout** on page 97).

Returns

Returns the data string received from SNA.

Throws

None.

Examples

```
(set! data(snalu0-recv hSNA 200 2000))
```

Location

```
monksnalua.monk
```

snalu0-get-property

Description

Obtains the property from the previous send or receive call.

Signature

```
(snalu0-get-property snaHandle propertyName)
```

Parameters

Name	Type	Description
snaHandle	Opaque handle	The handle to the SNA Server.
propertyName	String	SNA Property types: ▪ lua_th ▪ lua_rh ▪ lua_flag1 ▪ lua_flag2 ▪ lua_message_type ▪ lua_inc_th_snf (returns current snf + 1) ▪ lua_th_snf (returns current sequence number)

Returns

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

Throws

None.

Examples

```
(snalu0-send hSNA "Hello There")
```

Location

```
monksnalua.monk
```

snalu0-set-property

Description

Sets the specified property in SNA, generally precedes the receive or send call.

Signature

```
(snalu0-set-property snaHandle propertyName propertyValue)
```

Parameters

Name	Type	Description
snaHandle	Opaque handle	The handle to the SNA Server.
propertyName	String	SNA Property types: ▪ lua_th ▪ lua_rh ▪ lua_flag1 ▪ lua_flag2 ▪ lua_message_type ▪ lua_inc_th_snf (returns current snf + 1) ▪ lua_th_snf (returns current sequence number)
propertyValue	String	Property value to set.

Returns

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

Throws

None.

Examples

```
(snalu0-send hSNA "Hello There")
```

Location

```
monksnalua.monk
```

8.3 Standard e*Way Functions

The functions described in this section control the SNA e*Way's communications center and are defined within the configuration file. None of these functions is available to Collaboration Rules scripts executed by the e*Way.

8.3.1 LU6.2

The SNA e*Way's standard Monk functions for LU6.2 are:

sna-init on page 113

sna-conn-establish on page 114

sna-conn-verify on page 114

sna-conn-shutdown on page 115

sna-incoming on page 115

sna-outgoing on page 116

sna-pos-ack on page 117

sna-neg-ack on page 117

sna-shutdown on page 118

sna-startup on page 119

sna-init

Description

Begins the initialization process for the e*Way. This function loads the **stc_monksna.dll** file and the initialization file, thereby making the function scripts available for future use.

Signature

```
(sna-init)
```

Parameters

None.

Returns

The string "FAILURE" causes the e*Way to shut down. Any other return indicates success.

Throws

None.

Additional Information

Within this function, any necessary global variables to be used by the function scripts could be defined. The internal function that loads this file is called once when the e*Way first starts up.

See **Monk Environment Initialization File** on page 71 for more information.

Location

```
sna-init.monk
```

sna-conn-establish

Description

Establishes a connection to the external system.

Signature

```
(sna-conn-establish)
```

Parameters

None.

Returns

The string “UP” indicates the connection was established successfully. Anything else indicates no connection.

Throws

None.

Additional Information

See **External Connection Establishment Function** on page 74 for more information.

Location

```
sna-conn-establish.monk
```

sna-conn-verify

Description

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

Signature

```
(sna-conn-verify)
```

Parameters

None.

Returns

The string “UP” indicates the connection is currently established. Anything else indicates no connection.

Throws

None.

Additional Information

See **External Connection Verification Function** on page 75 and on page 93 for more information.

Location

```
sna-conn-verify.monk
```

sna-conn-shutdown

Description

Requests that the external connection shut down.

Signature

```
(sna-conn-shutdown shutdown)
```

Parameters

Name	Type	Description
shutdown	string	The function that passes the string "SUSPEND_NOTIFICATION" to the external system before the e*Way shuts down.

Returns

The string "SUCCESS" indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed until a **shutdown-request** call is executed successfully.

Throws

None.

Additional Information

If a return value of "SUCCESS" is not returned, then you must execute a **shutdown-request** call from within a Monk function to allow the requested shutdown to process to continue.

See **External Connection Shutdown Function** on page 75 for more information.

Location

```
sna-conn-shutdown.monk
```

sna-incoming

Description

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

Signature

```
(sna-incoming)
```

Parameters

None.

Returns

- An empty string indicates a successful operation, but nothing is sent to e*Gate.
- A string containing Event data indicates successful operation, and the returned Event is sent to e*Gate.
- The string "CONNERR" indicates a problem with the connection to the external system. When the connection is re-established this function is re-executed with the same input Event.

Throws

None.

Additional Information

See **Exchange Data with External Function** on page 73 for more information.

Location

```
sna-incoming.monk
```

sna-outgoing

Description

Sends a received Event from e*Gate to the external system.

Signature

```
(sna-outgoing event-string)
```

Parameters

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

Returns

- An empty string indicates a successful operation.
- The string "RESEND" causes the Event to be immediately resent.
- The string "CONNERR" indicates a problem with the connection to the external system. When the connection is re-established this function is re-executed with the same input Event.
- The string "DATAERR" indicates the function had a problem processing data. If the e*Gate journal is enabled, the Event is journaled and the failed Event count is increased. (The input Event is essentially skipped in this process.) Use the **event-send-to-egate** function to place bad Events in a bad Event queue.

Throws

None.

Additional Information

See **Process Outgoing Message Function** on page 72 for more information.

Location

```
sna-outgoing.monk
```

sna-pos-ack

Description

Sends a positive acknowledgment to the external system after all Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Signature

```
(sna-pos-ack arg)
```

Parameters

Name	Type	Description
arg	string	The Event for which an acknowledgment is sent.

Returns

- An empty string indicates a successful operation. The e*Way is then be able to proceed with the next request.
- The string “CONNERR” indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.

Additional Information

See **Positive Acknowledgment Function** on page 76 for more information.

Location

```
sna-pos-ack.monk
```

sna-neg-ack

Description

Sends a negative acknowledgment to the external system when the e*Way fails to process and queue Events from the external system.

Signature

```
(sna-neg-ack arg)
```

Parameters

Name	Type	Description
arg	string	The Event for which a negative acknowledgment is sent.

Returns

- An empty string indicates a successful operation.
- The string “CONNERR” indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.

Throws

None.

Additional Information

See **Negative Acknowledgment Function** on page 76 for more information.

Location

```
sna-neg-ack.monk
```

sna-shutdown

Description

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

Signature

```
(sna-shutdown command)
```

Parameters

Name	Type	Description
command	string	The function that passes the string "SHUTDOWN_NOTIFICATION" to the external system before the e*Way shuts down.

Returns

Returns a null string.

Throws

None.

Additional Information

See **Shutdown Command Notification Function** on page 77 for more information.

Location

```
sna-shutdown.monk
```

sna-startup

Description

Invokes startup and is used for function loads that are specific to this e*Way.

Signature

```
(sna-startup)
```

Parameters

None.

Returns

The string "FAILURE" causes the e*Way to shut down. Any other return indicates success.

Throws

None.

Additional Information

This function should be used to initialize the external system before data exchange starts. Any additional variables may be defined here.

See **Startup Function** on page 72 for more information.

Location

```
sna-startup.monk
```

8.3.2 LUA

The SNA e*Way's standard Monk functions for LUA (and LU0) are:

snalu0-init on page 120

snalu0-conn-establish on page 121

snalu0-conn-verify on page 121

snalu0-conn-shutdown on page 121

snalu0-incoming on page 122

snalu0-outgoing on page 123

snalu0-pos-ack on page 123

snalu0-neg-ack on page 124

snalu0-shutdown on page 124

snalu0-startup on page 125

snalu0-init

Description

Begins the initialization process for the e*Way. This function loads the **stc_monksnalu0.dll** file and the initialization file, thereby making the function scripts available for future use.

Signature

```
(snalu0-init)
```

Parameters

None.

Returns

The string "FAILURE" causes the e*Way to shut down. Any other return indicates success.

Throws

None.

Additional Information

Within this function, any necessary global variables to be used by the function scripts could be defined. The internal function that loads this file is called once when the e*Way first starts up.

See **Monk Environment Initialization File** on page 89 for more information.

snalu0-conn-establish

Description

Establishes a connection to the external system.

Signature

```
(snalu0-conn-establish)
```

Parameters

None.

Returns

The string "UP" indicates the connection was established successfully. Anything else indicates failure to connect.

Throws

None.

Additional Information

See **External Connection Establishment Function** on page 92 for more information.

snalu0-conn-verify

Description

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

Signature

```
(snalu0-conn-verify)
```

Parameters

None.

Returns

The string "UP" if connection established. Any other value indicates the connection is not established.

Throws

None.

Additional Information

See **External Connection Verification Function** on page 93 for more information.

snalu0-conn-shutdown

Description

Requests that the external connection shut down. A return value of "SUCCESS" indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a (**shutdown-request** on page 131) call from within a Monk function to allow the requested shutdown to process to continue.

Signature

```
(snalu0-conn-shutdown shutdown)
```

Parameters

Name	Type	Description
shutdown	string	The function that passes the string "SUSPEND_NOTIFICATION" to the external system before the e*Way shuts down.

Returns

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

Throws

None.

Additional Information

See **External Connection Shutdown Function** on page 93 for more information.

snalu0-incoming

Description

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

Signature

```
(snalu0-incoming)
```

Parameters

None.

Returns

- An empty string indicates a successful operation, but nothing is sent to e*Gate.
- A string containing Event data indicates successful operation, and the returned Event is sent to e*Gate.
- The string **"CONNERR"** indicates a problem with the connection to the external system. When the connection is re-established this function is re-executed with the same input Event.

Throws

None.

Additional Information

See **Exchange Data with External Function** on page 91 for more information.

snalu0-outgoing

Description

Sends a received Event from e*Gate to the external system.

Signature

```
(snalu0-outgoing event-string)
```

Parameters

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

Returns

- An empty string indicates a successful operation.
- The string "RESEND" causes the Event to be immediately resent.
- The string "CONNERR" indicates a problem with the connection to the external system. When the connection is re-established this function is re-executed with the same input Event.
- The string "DATAERR" indicates the function had a problem processing data. If the e*Gate journal is enabled, the Event is journaled and the failed Event count is increased. (The input Event is essentially skipped in this process.) Use the **event-send-to-egate** function to place bad events in a bad event queue.

Throws

None.

Additional Information

See **Process Outgoing Message Function** on page 90 for more information.

snalu0-pos-ack

Description

Sends a positive acknowledgment to the external system after all Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Signature

```
(snalu0-pos-ack arg)
```

Parameters

Name	Type	Description
arg	string	The Event for which an acknowledgment is sent.

Returns

- An empty string indicates a successful operation. The e*Way is then be able to proceed with the next request.
- The string "CONNERR" indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.

Additional Information

See **Positive Acknowledgment Function** on page 94 for more information.

snalu0-neg-ack

Description

Sends a negative acknowledgment to the external system when the e*Way fails to process and queue Events from the external system.

Signature

```
(snalu0-neg-ack arg)
```

Parameters

Name	Type	Description
arg	string	The Event for which a negative acknowledgment is sent.

Returns

An empty string indicates a successful operation.

The string "CONNERR" indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.

Throws

None.

Additional Information

See **Negative Acknowledgment Function** on page 94 for more information.

snalu0-shutdown

Description

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

Signature

```
(snalu0-shutdown command)
```

Parameters

Name	Type	Description
command	string	The function that passes the string "SHUTDOWN_NOTIFICATION" to the external system before the e*Way shuts down.

Returns

Returns a null string.

Throws

None.

Additional Information

See **Shutdown Command Notification Function** on page 95 for more information.

snalu0-startup

Description

Invokes startup and is used for function loads that are specific to this e*Way.

Signature

```
(snalu0-startup)
```

Parameters

None.

Returns

The string “FAILURE” causes the e*Way to shut down. Any other return indicates success.

Throws

None.

Additional Information

This function should be used to initialize the external system before data exchange starts. Any additional variables may be defined here.

See **Startup Function** on page 90 for more information.

8.4 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 126

event-rollback-to-egate on page 127

event-send-to-egate on page 127

event-send-to-egate-ignore-shutdown on page 128

event-send-to-egate-no-commit on page 128

get-logical-name on page 129

insert-exchange-data-event on page 129

send-external-up on page 130

send-external-down on page 130

shutdown-request on page 131

start-schedule on page 131

stop-schedule on page 132

waiting-to-shutdown on page 132

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 128

event-send-to-egate-no-commit on page 128

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 127

event-send-to-egate-no-commit on page 128

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 126

event-rollback-to-egate on page 127

event-send-to-egate on page 127

event-send-to-egate-ignore-shutdown on page 128

get-logical-name

Description

Returns the logical name of the e*Way.

Signature

```
(get-logical-name)
```

Parameters

None.

Returns

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

Throws

None.

insert-exchange-data-event

Description

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

Signature

```
(insert-exchange-data-event)
```

Parameters

None.

Returns

None.

Throws

None.

See also

Exchange Data with External Function on page 73 (LU6.2) or **Exchange Data with External Function** on page 91 (LUA/LU0).

Exchange Data Interval on page 67 (LU6.2) or **Exchange Data Interval** on page 85 (LUA/LU0).

Zero Wait Between Successful Exchanges on page 67 (LU6.2) or **Zero Wait Between Successful Exchanges** on page 85 (LUA/LU0).

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.

See also

Shutdown Command Notification Function on page 77 (LU6.2) or **Shutdown Command Notification Function** on page 95 (LUA/LU0).

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.

See also

Exchange Data with External Function on page 73 (LU6.2) or **Exchange Data with External Function** on page 91 (LUA/LU0).

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.

See also

Exchange Data with External Function on page 73 (LU6.2) or **Exchange Data with External Function** on page 91 (LUA/LU0).

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

A

Ack String parameter **80**
Additional Path parameter **71**, **89**
APIs - see functions, Monk
Assigning ETDs to Event Types **32**
Autorun **21**
Auxiliary Library Directories parameter **71**, **89**

C

Changing the User Name **49**
Collaboration **34**, **53**, **56**
  Rules **56**
  Service **53**
Communication Setup - see configuration
configuration
  Communication Setup (LU0) **85–87**
  Communication Setup (LUA) **67–69**
  General Settings (LU0) **83–84**
  General Settings (LUA) **65–66**
  Monk Configuration (LU0) **88–96**
  Monk Configuration (LUA) **70–78**
  SNA Client Configuration (LU0) **97**
  SNA Client Configuration (LUA) **79–81**
configuration parameters
  Ack String **80**
  Additional Path **71**, **89**
  Auxiliary Library Directories **71**, **89**
  Control Bytes **97**
  Data Flow **81**
  Down Timeout **69**, **87**
  Exchange Data Interval **67**, **85**
  Exchange Data With External Function **73**, **91**
  External Connection Establishment Function **74**, **92**
  External Connection Shutdown Function **75**, **93**
  External Connection Verification Function **75**, **93**
  Forward External Errors **84**
  Forward External Errors (LUA) **66**
  Initialize Conversation **81**
  Journal File Name **83**
  Journal File Name (LUA) **65**
  Local LU Name **79**, **97**
  Max Failed Messages **83**
  Max Failed Messages (LUA) **65**
  Max Message Name **97**
  Max Resends Per Message **83**
  Max Resends Per Message (LUA) **65**
  Monk Environment Initialization File **71**, **89**
  Nak String **80**
  Negative Acknowledgment Function **76**, **94**
  PacketSize **79**
  Positive Acknowledgement Function **76**, **94**
  Process Outgoing Message Function **72**, **90**
  Receive Timeout **97**
  Request Reply **80**
  Resend Timeout **69**, **87**
  Shutdown Command Notification Function **77**, **95**
  Start Exchange Data Schedule **68**, **69**, **86**, **87**
  Startup Function **72**, **90**
  Stop Exchange Data Schedule **69**, **86**
  SYMDESTNAME **79**
  Synchronization Level **81**
  Timeout **80**
  Up Timeout **69**, **87**
  Use Ack Nak **80**
  Zero Wait Between Successful Exchanges **67**, **85**
configuration procedures **45**
Control Bytes parameter **97**
conventions, writing in document **8**

D

Data Flow parameter **81**
Down Timeout parameter **69**, **87**

E

e*Way
  configuration **45**
  creating **43**
  Installation **21**
  Properties **44**
  Schedules **49**
  Startup Options **49**
  troubleshooting **53**
Editor
  Collaboration Rules **56**
Event Type **32**
Event Type Definition (ETD) **32**
event-commit-to-egate function **126**
event-rollback-to-egate function **127**
Events **55**
event-send-to-egate function **127**
event-send-to-egate-ignore-shutdown function **128**
event-send-to-egate-no-commit function **128**
Exchange Data Interval parameter **67**, **85**

Exchange Data with External Function parameter 73, 91
External Connection Establishment Function parameter 74, 92
External Connection Shutdown Function parameter 75, 93
External Connection Verification Function parameter 75, 93

F

Forward External Errors (LUA) parameter 66
Forward External Errors parameter 84
functions (see also functions, Monk)
  Generic 126–133
  Native 99–112
  Standard 113–125
functions, Monk
  event-commit-to-egate 126
  event-rollback-to-egate 127
  event-send-to-egate 127
  event-send-to-egate-ignore-shutdown 128
  event-send-to-egate-no-commit 128
  get-logical-name 129
  insert-exchange-data-event 129
  send-external down 130
  send-external-up 130
  shutdown-request 131
  sna-accept-conversation 99
  sna-change-state 100
  sna-change-state-no-synch 100
  sna-client-connect 102
  sna-client-connect-no-synch 102
  sna-client-disconnect 103
  sna-client-isconnected 103
  sna-client-recv 104
  sna-client-recv-no-synch 105
  sna-client-send 105
  sna-client-send-no-synch 106
  sna-confirmed 101
  sna-conn-establish 114
  sna-conn-shutdown 115
  sna-conn-verify 114
  sna-incoming 115
  sna-init 113
  snalu0-connect 108
  snalu0-conn-establish 121
  snalu0-conn-shutdown 121
  snalu0-conn-verify 121
  snalu0-disconnect 108
  snalu0-get-property 111
  snalu0-incoming 122
  snalu0-init 120
  snalu0-isconnected 109
  snalu0-neg-ack 124
  snalu0-outgoing 123
  snalu0-pos-ack 123
  snalu0-recv 110
  snalu0-send 110
  snalu0-set-property 111
  snalu0-shutdown 124
  sna-neg-ack 117
  sna-outgoing 116
  sna-pos-ack 117
  sna-shutdown 118
  sna-startup 119, 125
  start-schedule 131
  stop-schedule 132
  waiting-to-shutdown 132

G

General Settings - see configuration
Generic e*Way Functions 126–133
get-logical-name function 129

I

Initialize Conversation parameter 81
insert-exchange-data-event function 129
Installation procedures
  e*Way (UNIX) 24
  e*Way (Windows) 21
InstallShield 21
Intelligent Queue (IQ) 35, 53

J

Journal File Name (LUA) parameter 65
Journal File Name parameter 83

L

Load Path, Monk 70, 88
Local LU Name parameter 79, 97
logging options 51

M

Max Failed Messages (LUA) parameter 65
Max Failed Messages parameter 83
Max Message Name parameter 97
Max Resends Per Message (LUA) parameter 65
Max Resends Per Message parameter 83
monitoring thresholds 52
Monk Configuration
  Load Path 70, 88

Specifying File Names **70**, **88**
Specifying Function Names **70**, **88**
Specifying Multiple Directories **70**, **88**
Monk Configuration - see configuration
Monk Environment Initialization File parameter **71**, **89**
Monk functions - see functions, Monk

N

Nak String parameter **80**
Native e*Way Functions - see functions
Negative Acknowledgment Function parameter **76**, **94**

P

PacketSize parameter **79**
parameters - see configuration parameters
Participating Host **53**
Patch Requirements, Solaris **19**
Positive Acknowledgment Function parameter **76**, **94**
procedures
configuration **45**
installation **21**
Process Outgoing Message Function parameter **72**, **90**
Properties, e*Way **44**
publish **53**

Q

Queue - see Intelligent Queue (IQ)

R

Receive Timeout parameter **97**
Request Reply parameter **80**
Resend Timeout parameter **69**, **87**

S

Schedules **49**
send-external down function **130**
send-external-up function **130**
Setting Startup Options or Schedules **49**
Shutdown Command Notification Function parameter **77**, **95**
shutdown-request function **131**
SNA Client Configuration **79**
SNA Client Configuration - see configuration
sna-accept-conversation function **99**
sna-change-state function **100**
sna-change-state-no-synch function **100**
sna-client-connect function **102**
sna-client-connect-no-synch function **102**
sna-client-disconnect function **103**
sna-client-isconnected function **103**
sna-client-recv function **104**
sna-client-recv-no-synch function **105**
sna-client-send function **105**
sna-client-send-no-synch function **106**
sna-confirmed function **101**
sna-conn-establish function **114**
sna-conn-shutdown function **115**
sna-conn-verify function **114**
sna-incoming function **115**
sna-init function **113**
snalu0-connect function **108**
snalu0-conn-establish function **121**
snalu0-conn-shutdown function **121**
snalu0-conn-verify function **121**
snalu0-disconnect function **108**
snalu0-get-property function **111**
snalu0-incoming function **122**
snalu0-init function **120**
snalu0-isconnected function **109**
snalu0-neg-ack function **124**
snalu0-outgoing function **123**
snalu0-pos-ack function **123**
snalu0-recv function **110**
snalu0-send function **110**
snalu0-set-property function **111**
snalu0-shutdown function **124**
sna-neg-ack function **117**
sna-outgoing function **116**
sna-pos-ack function **117**
sna-shutdown function **118**
sna-startup function **119**, **125**
Solaris Patch Requirements **19**
Standard e*Way Functions - see functions
Start Exchange Data Schedule parameter **68**, **69**, **86**, **87**
start-schedule function **131**
Startup Function parameter **72**, **90**
Startup Options **49**
Stop Exchange Data Schedule parameter **69**, **86**
stop-schedule function **132**
subscribe **53**
SYMDESTNAME parameter **79**
Synchronization Level parameter **81**

T

Timeout parameter **80**
troubleshooting the e*Way **53**

U

UNIX installation procedures **24**
Up Timeout parameter **69**, **87**
Use Ack Nak parameter **80**
User name **49**

W

waiting-to-shutdown function **132**
Windows installation procedures **21**

Z

Zero Wait Between Successful Exchanges parameter **67**, **85**