SeeBeyond ICAN Suite

e*Way Intelligent Adapter for BroadVision User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

SEEBEYOND®

Version 20050405212542.

Contents

Preface

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

P.1 Intended Reader

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

- Windows 2000 and/or UNIX operations and administration
- Windows-style GUI operations
- BroadVision One-To-One applications

P.2 Organization

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

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

P.3 Nomenclature

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

P.4 Online Viewing

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

Hyperlinking is available in:

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

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

P.5 Writing Conventions

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

Monospaced (Courier) Font

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

```
Configuration for BOB_Promotion

java -jar ValidationBuilder.jar
```

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

```
stcregutil -rh <host-name> -un <user-name> -up <password> -sf
```

Bold Sans-serif Font

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

Chapter 1

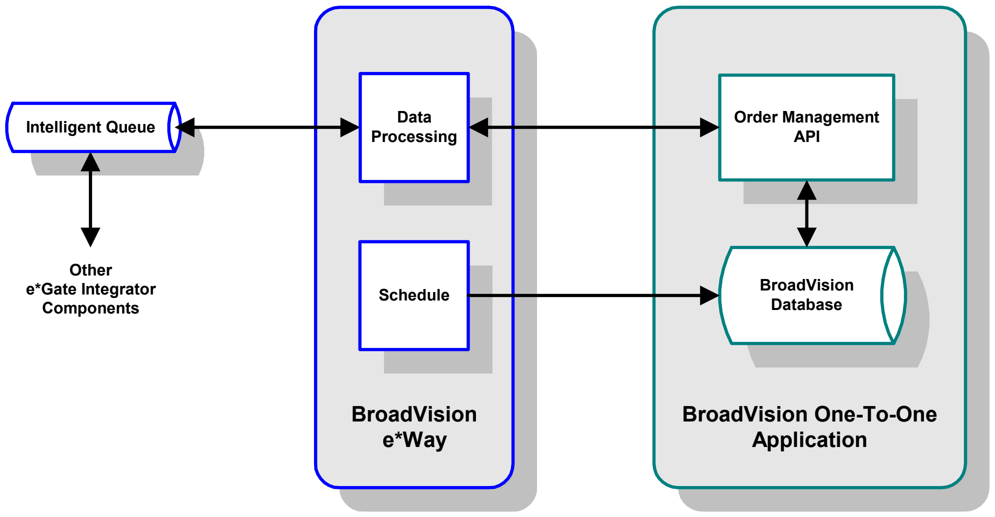
Introduction

This chapter provides a brief introduction to the SeeBeyond e*Way Intelligent Adapter for BroadVision.

1.1 Overview

The e*Way Intelligent Adapter for BroadVision enables the e*Gate system to exchange data with BroadVision One-To-One business applications. The e*Way can operate in either inbound-to or outbound-from-BroadVision mode, at near-real-time speed.

Figure 1 BroadVision e*Way Process Flow

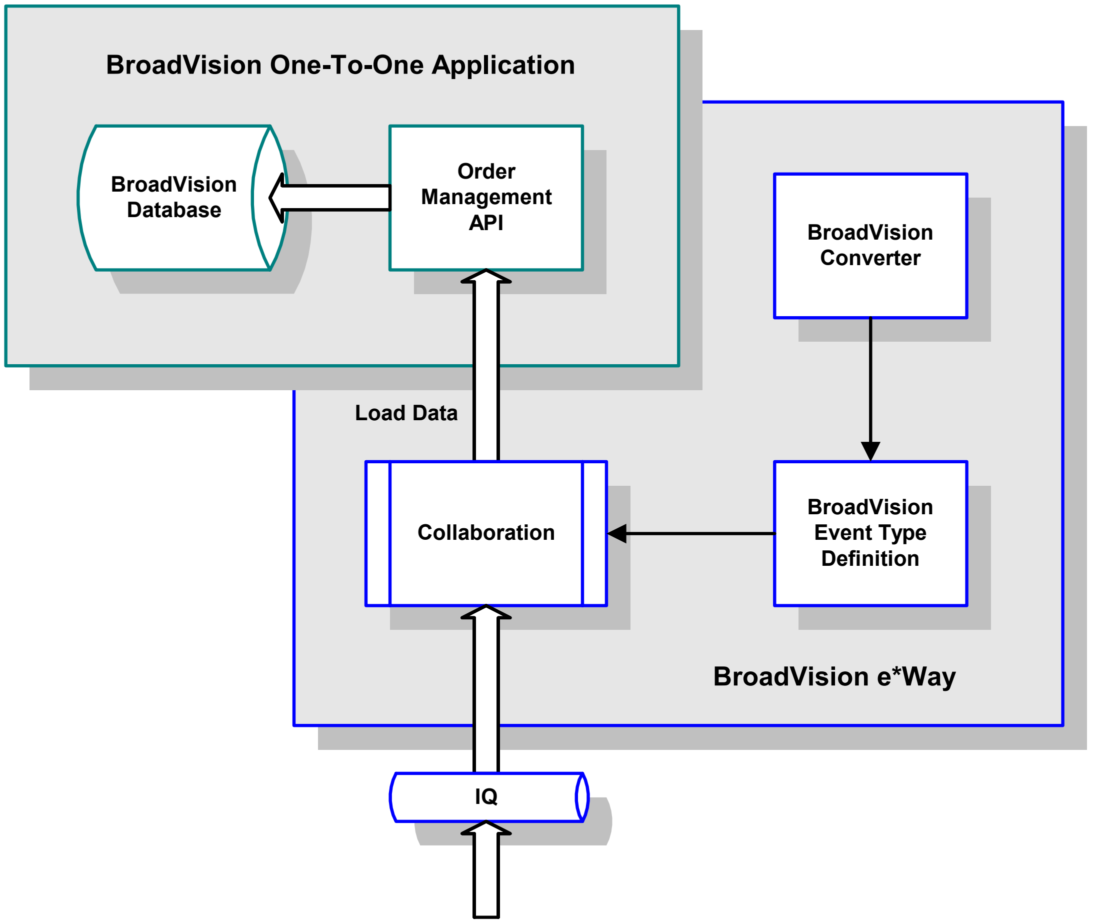


The BroadVision e*Way interacts with BroadVision's Order Management API to pass data to and from the BroadVision database. When operating in BroadVision-outbound mode, the e*Way polls the BroadVision database directly according to a user-configured schedule. By defining the polling interval to be very short (e.g., seconds), the response approximates that of an event-driven system.

1.2 Interaction with BroadVision

1.2.1 e*Gate to BroadVision

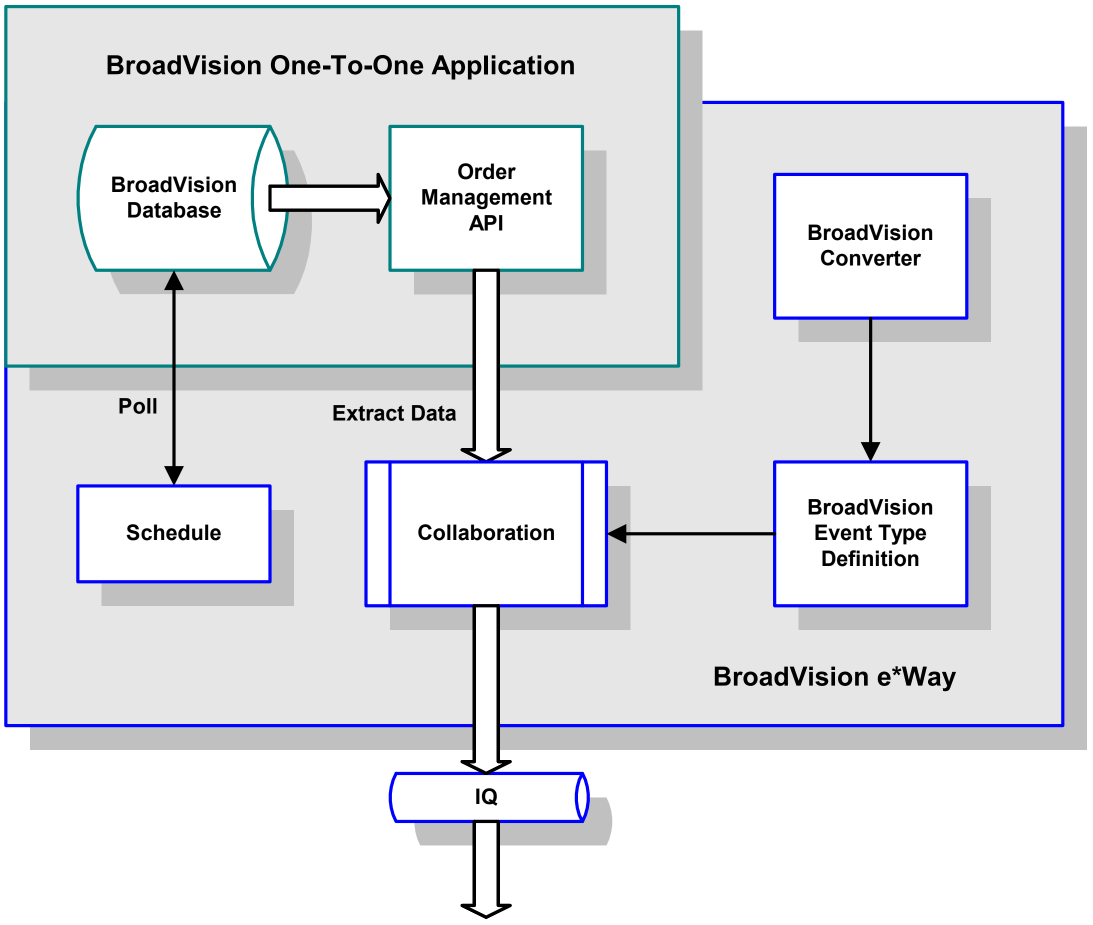
Figure 2 e*Gate-to-BroadVision Process Flow



1. The BroadVision e*Way extracts data from an Intelligent Queue for processing.
2. The e*Way processes the information following a Collaboration incorporating an ETD created previously using the BroadVision Converter.
3. The e*Way loads data into the BroadVision database by means of the BroadVision Order Management API.

1.2.2 BroadVision to e*Gate

Figure 3 BroadVision-to-e*Gate Process Flow

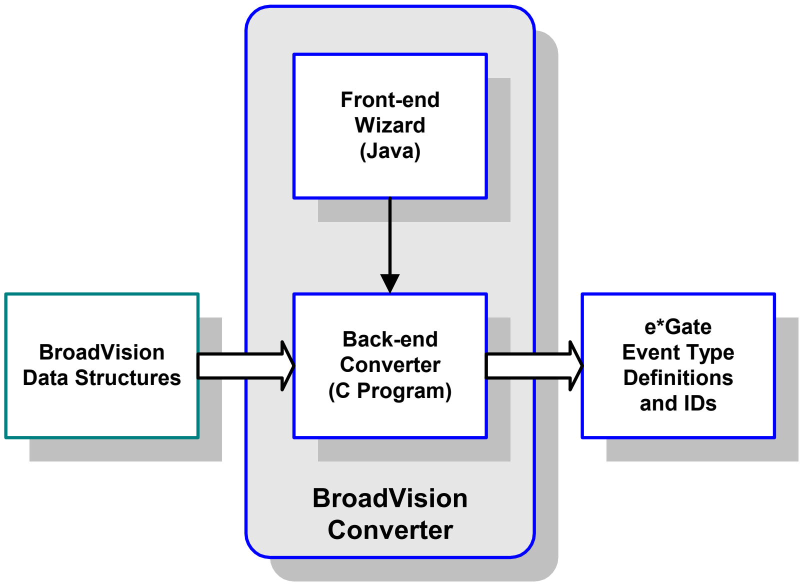


1. The BroadVision e*Way polls the BroadVision database according to a predefined schedule.
2. The e*Way extracts new data from BroadVision by means of the BroadVision Order Management API.
3. The e*Way processes the information following a Collaboration incorporating an ETD created previously using the BroadVision Converter.
4. The e*Way sends the processed data to an Intelligent Queue for further processing and/or routing to another application.

1.3 The BroadVision Converter

Event Type Definitions (and ultimately Collaborations) are prepared using the BroadVision Converter, which is integrated with the ETD Editor. The BroadVision Converter has two basic components: the Java Wizard (front end) and the BroadVision C-based converter (back end). For the majority of installations, the front end is on a 2000 platform and the back end—which has to be on the same host as the BroadVision web server—is usually on a UNIX machine.

Figure 4 BroadVision Converter


Front-end Wizard (Java)
BroadVision Data Structures
Back-end Converter (C Program)
e*Gate Event Type Definitions and IDs
BroadVision Converter

Java's Remote Method Invocation (RMI) is used to invoke the back-end converter remotely. RMI also has two parts: the Java RMI registry (**rmiregistry.exe**) and the required BV Java RMI Converter Server classes.

For Windows, both parts are run as services, and are installed automatically by InstallShield. For UNIX, however, you need to add a file to the system initialization directory manually, and also modify some environment settings. Instructions are found in **Environment Configuration** on page 23.

1.4 e*Way Components

The BroadVision e*Way incorporates the following:

- An executable file, **stcewgenericmonk.exe**, installed as part of e*Gate Integrator
- Dynamic load libraries, **stc_ewbv.dll** and **stc_ewbv55.dll**, which extend the Generic e*Way Kernel to form the BroadVision e*Way
- A default configuration file, **ewbv.def**
- Monk function scripts and library files, discussed in **Chapter 7**
- The BroadVision Converter, **stcbvconvert.exe** and **stcbv55convert.exe**, used to build Event Type Definitions
- Example schema, discussed in **Sample Schemas** on page 35

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

1.5 Supported Operating Systems

The e*Way Intelligent Adapter for BroadVision is available on the following s:

- Windows 2000 and Windows Server 2003
- HP-UX 11.0
- Sun Solaris 8
- Japanese Windows 2000 and Windows Server 2003
- Japanese HP-UX 11.0
- Japanese Sun Solaris 8

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

Chapter 2

Installation

This chapter describes the requirements and procedures for installing the e*Way Intelligent Adapter for BroadVision. Once you have installed this e*Way, you must configure it for your system and incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types. See **Chapter 3** for information on implementing a working system.

2.1 System Requirements

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

1. An e*Gate Participating Host, version 5.0 or later.
2. A TCP/IP network connection
3. Sufficient free disk space to accommodate e*Way files:
   - Approximately 15 MB on Windows systems
   - Approximately 28 MB on Solaris systems
   - Approximately 19 MB on HP-UX systems

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

Note: *The BroadVision e*Way must be installed on the BroadVision system host computer.*

2.1.1 External System Requirements

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

- BroadVision One-To-One Enterprise 4.1 or 5.5

Note: *The BroadVision RMI Registry requires jre 1.3 or later.*

2.2 Installing the e*Way

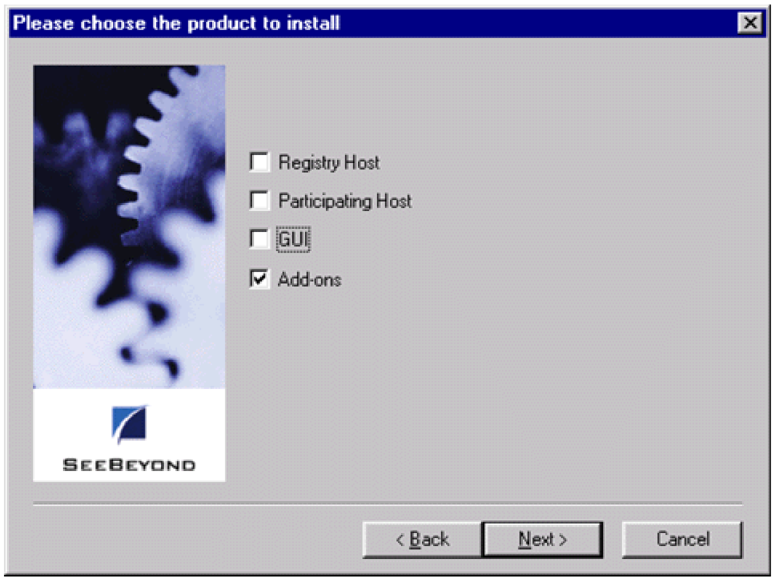
2.2.1 Windows Systems

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

To install the e*Way on a Windows 2000 system

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Exit all Windows programs and disable any anti-virus applications before running the setup program.
3. Insert the e*Way installation CD-ROM into the CD-ROM drive.
4. If the CD-ROM drive's Autorun feature is enabled, the setup application should launch automatically. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
5. The InstallShield setup application launches. Follow the on-screen instructions until you come to the **Choose Product** screen.

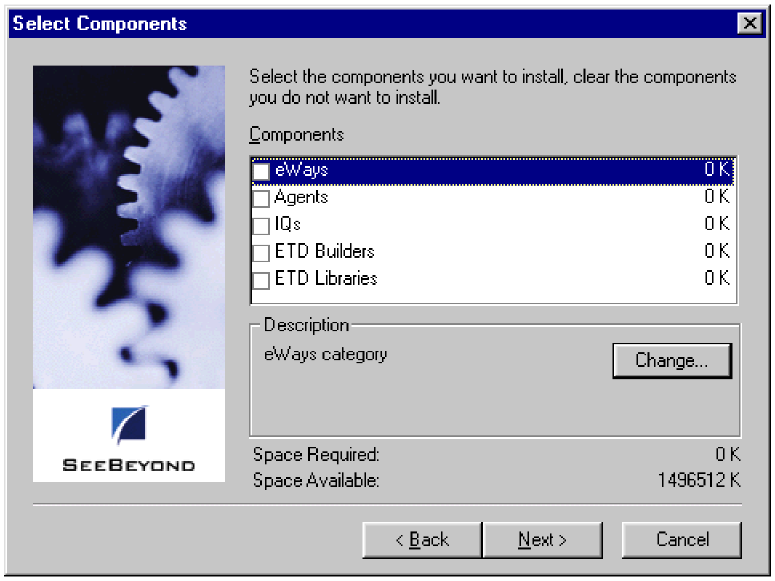
Figure 5 Choose Product Dialog



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

6. Check **Add-ons**, then click **Next**. Again follow the on-screen instructions.
7. When the **Select Components** dialog box appears, highlight—but do not check—**eWays** and then click **Change**.

Figure 6 Select Components Dialog



Select Components

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

Components

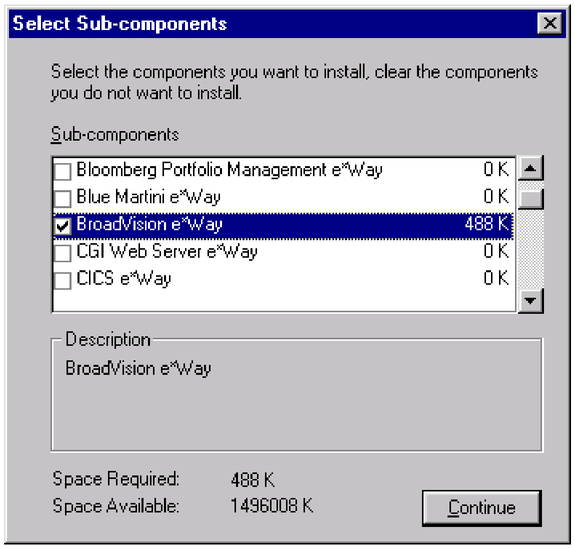
☐ eWays	0 K
☐ Agents	0 K
☐ IQs	0 K
☐ ETD Builders	0 K
☐ ETD Libraries	0 K

Description
eWays category

Space Required: 0 K
Space Available: 1496512 K

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

Figure 7 Select e*Way Dialog



Select Sub-components

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

Sub-components

☐ Bloomberg Portfolio Management e*Way	0 K
☐ Blue Martini e*Way	0 K
☑ BroadVision e*Way	488 K
☐ CGI Web Server e*Way	0 K
☐ CICS e*Way	0 K

Description
BroadVision e*Way

Space Required: 488 K
Space Available: 1496008 K

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

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

Subdirectories and Files

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

Table 1 Participating Host & Registry Host

Subdirectories	Files
\bin\	stc_ewbv.dll stc_ewbv55.dll stcbvconvert.exe stcbv55convert.exe stcsvcinstd.exe
\configs\stcewgenericmonk\	ewbv.def
\monk_library\	ewbv.gui
\monk_library\ewbv\	bv-category-create.monk bv-category-delete.monk bv-category-get-cat-entry.monk bv-category-move.monk bv-category-rename.monk bv-cnt-delete.monk bv-cnt-get-productname.monk bv-cnt-internal.monk bv-cnt-sql-select.monk bv-cnt-struct-create.monk bv-cnt-struct-update.monk bv-content-ref-create.monk bv-content-ref-delete.monk bv-content-ref-list.monk bv-order-complete-fulfill.monk bv-order-get-accountname.monk bv-order-get-ordernumber.monk bv-order-get-orderprop-ordernum.monk bv-order-get-orders.monk bv-order-get-useralias.monk bv-order-get-userid.monk bv-order-internal.monk bv-order-partial-fulfill.monk bv-order-set-state.monk bv-order-struct-create.monk bv-order-struct-update.monk bv-util.monk bv.monk ewbv-init.monk ewbv-shutdown.monk

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

Table 2 Registry Host Only

Subdirectories	Files
\	stcewbv.ctl

Environment Configuration

The BroadVision starting process automatically appends the directories **bv1to1\bin** and **bv1to1\orbix\bin** to the Path environment variable.

For the BroadVision e*Way to run as a Windows 2000 Service, any environment variables required by BroadVision should be set at the System level. These include:

BV_DB_DATABASE	ORACLE_HOME
BV_DB_USER	ORACLE_SID
BV_DB_PASSWD	IT_DAEMON_PORT
BV_DB_SERVER	BV1TO1
BV_DB_VENDOR	BV1TO1_VAR

Follow the path **Start** > **Settings** > **Control Panel** > **System** > **Environment** to display the System variables settings.

For the BroadVision Converter, specific settings variables need to be modified according to the your environment on the Participating Host, as shown in Table 3, below.

Table 3 Settings Variables Requiring Modification

Variable	Description
BVUSER	BroadVision user name
EGATEDIR	Base directory of e*Gate installation
JREHOME	Base directory of Java Runtime Environment (JRE) installation

Note: *See also* **Starting and Running the e*Way** *on page 51.*

2.2.2 UNIX Systems

Note: *You do not need root privileges to install this e*Way, but you do to set up the RMI service. You can log in under the user name that you wish to own the e*Way files, if the user has sufficient privilege to create files in the e*Gate directory tree.*

To install the BroadVision e*Way on a UNIX system:

1. Log in on 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, displaying several options. Select the **Install e*Way** option, and follow any additional on-screen instructions.

Note: *Be sure to install the e*Way and BroadVision Converter files in the* **BVUSER** *directory on the Participating Host.*

6. The file **S99stcsvcinstdx** can be found on the installation CD under the **utils\bvstaging\** subdirectory. This file should be copied into the appropriate host directory, as listed in Table 4, below. All else in the subdirectory should be left untouched.

Table 4 S99stcsvcinstdx Location

HPUX 11	Others
/sbin/rc3.d/	/etc/rc3.d/

To set up the RMI service

1. Copy the **egate.jar** and the **stcjcs.jar** file from an e*Gate GUI installation. This file is located in the **egate\client\classes** folder.
2. Copy the **egate.jar** and the **stcjcs.jar** file to an **egate\client\classes** folder on your Participating Host machine.
3. Modify **S99stcsvcinstdx** to have the correct entries, for example:

   ```
   BVUSER=bv
   EGATEDIR=/export/home/bv/egate/client
   JREHOME=/opt/Java/JDK-1_1
   JEXE=jre
   ```
4. Start the service by typing the following at the command line:

   ```
   S99stcsvcinstdx start
   ```
5. If the service has been started correctly you should see the message:

```
RMI service <BVtoETD> is available and the rmi process should be
running.
```

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 5 Participating Host & Registry Host

Subdirectories	Files
/	S99stcsvcinstdx instbv.sh
/bin/	stc_ewbv.dll stc_ewbv55.dll stcbvconvert stcbv55convert
/configs/stcewgenericmonk/	ewbv.def
/monk_library/	ewbv.gui
/monk_library/ewbv/	bv-category-create.monk bv-category-delete.monk bv-category-get-cat-entry.monk bv-category-move.monk bv-category-rename.monk bv-cnt-delete.monk bv-cnt-get-productname.monk bv-cnt-internal.monk bv-cnt-sql-select.monk bv-cnt-struct-create.monk bv-cnt-struct-update.monk bv-content-ref-create.monk bv-content-ref-delete.monk bv-content-ref-list.monk bv-order-complete-fulfill.monk bv-order-get-accountname.monk bv-order-get-ordernumber.monk bv-order-get-orderprop-ordernum.monk bv-order-get-orders.monk bv-order-get-useralias.monk bv-order-get-userid.monk bv-order-internal.monk bv-order-partial-fulfill.monk bv-order-set-state.monk bv-order-struct-create.monk bv-order-struct-update.monk bv-util.monk bv.monk ewbv-init.monk ewbv-shutdown.monk

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

Table 6 Registry Host Only

Subdirectories	Files
/	stcewbv.ctl

Environment Configuration

The BroadVision starting process automatically appends the directories **bv1to1\bin** and **bv1to1\orbix\bin** to the Path environment variable.

For the BroadVision e*Way to run as a UNIX Service, any environment variables required by BroadVision should be set at the System level. These include:

BV_DB_DATABASE	ORACLE_HOME
BV_DB_USER	ORACLE_SID
BV_DB_PASSWD	IT_DAEMON_PORT
BV_DB_SERVER	BV1TO1
BV_DB_VENDOR	BV1TO1_VAR

For the BroadVision Converter, specific settings variables need to be modified according to the your environment on the Participating Host, as shown in Table 7, below.

Table 7 Settings Variables Requiring Modification

Variable	Description
BVUSER	BroadVision user name
EGATEDIR	Base directory of e*Gate installation
JREHOME	Base directory of Java Runtime Environment (JRE) installation

Note: *See also* **Starting and Running the e*Way** *on page 51.*

2.3 Optional Example Files

The installation CD contains two sample schemas, **BV_Orders_Post**, and **BV_Products**, located in the **samples\ewbv** directory. To use a schema, you must load it onto your system using the following procedure. See **Sample Schemas** on page 35 for descriptions of the sample schema and instructions regarding its use.

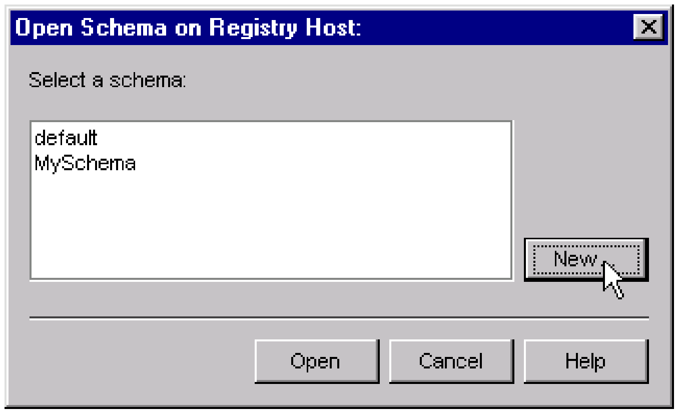
Note: *The BroadVision e*Way must be properly installed on your system before you can run the sample schema.*

2.3.1 Installation Procedure

To load a sample schema

1 Invoke the **Open Schema** dialog box and select **New** (see Figure 8).

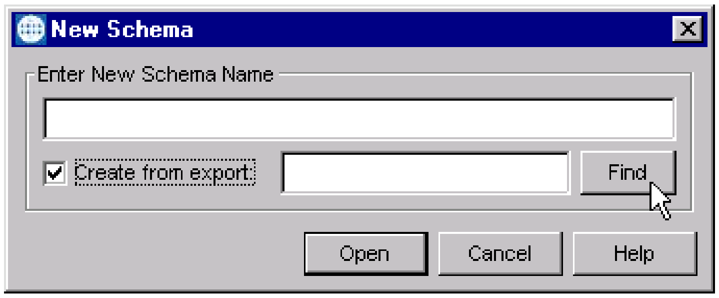
Figure 8 Open Schema Dialog



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

3 Select **Create from export** and navigate to the directory containing the sample schema by clicking the **Find** button (see Figure 9).

Figure 9 New Schema Dialog



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

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

2.3.2 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 8 Subdirectories and Files - BV_Orders_Post Schema

Subdirectories	Files
\	BV_Orders_Post.ctl
\runtime\configs\stcewfile\	orders_feeder.cfg orders_feeder.sc
\runtime\configs\stcewgenericmonk\	ewBVOrdersPost.cfg ewBVOrdersPost.sc
\runtime\data\Bv\	bv_orders.dat
\runtime\monk_scripts\common\	BV_Orders_Post.tsc BVSALESORDER.ssc

Table 9 Subdirectories and Files - BV_Products Schema

Subdirectories	Files
\	BV_Products.ctl
\runtime\configs\stcewfile\	feeder.cfg feeder.sc
\runtime\configs\stcewgenericmonk\	BV_Products.cfg BV_Products.sc
\runtime\data\Bv\	bv_products.dat
\runtime\monk_scripts\common\	BV_Products.ssc BV_Products.tsc BV_Products_Input.ssc

Chapter 3

System Implementation

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

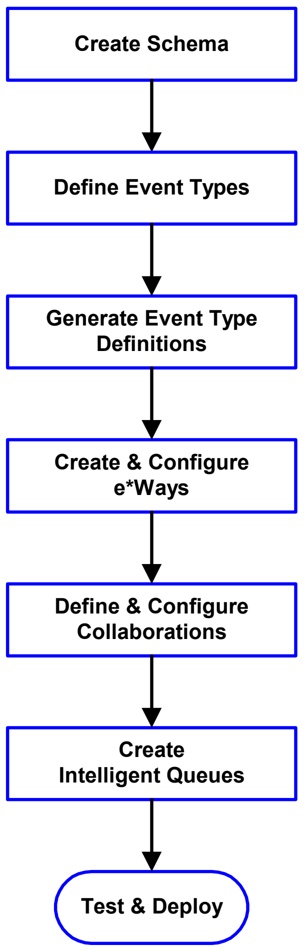
3.1 Overview

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

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

The topics discussed in this chapter include the following:

3.1.1 Implementation Sequence


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

1 The first step is to create a new Schema—the subsequent steps apply only to this Schema (see **Creating a Schema** on page 28).

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

3 Third, you need to associate the Event Types created in the previous step with Event Type Definitions (ETDs) derived from the applicable Business Rules (see **Creating Event Type Definitions** on page 30).

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.

3.1.2 Viewing e*Gate Components

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

3.2 Creating a Schema

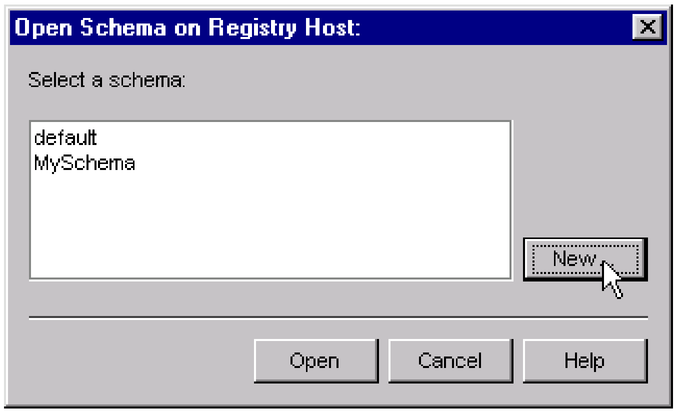
A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

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

To select or create a schema

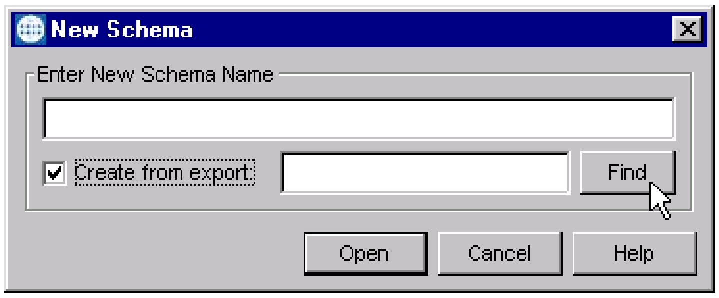
1 Invoke the **Open Schema** dialog box and **Open** an existing schema or click **New** to create a new schema.

Figure 10 Open Schema Dialog



2 Clicking **New** invokes the **New Schema** dialog box (Figure 11).

Figure 11 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

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

3.4.1 Using the ETD Editor's Build Tool

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

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

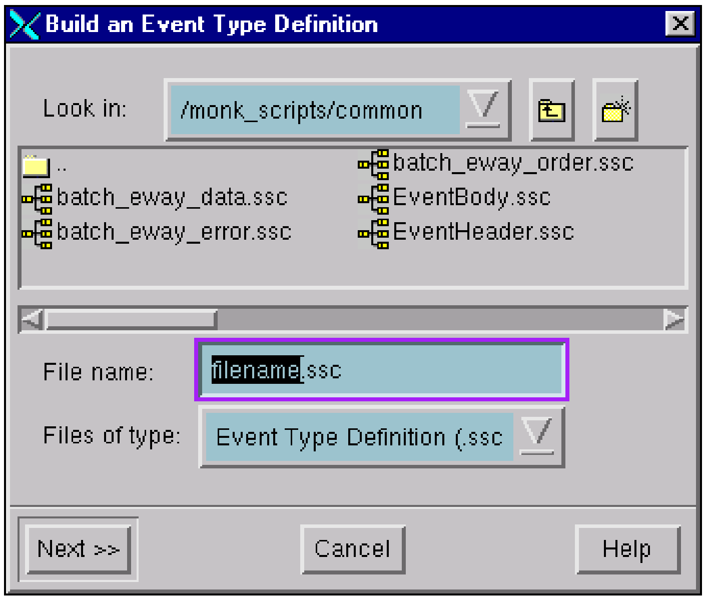
To create an Event Type Definition using the Build tool

1 Launch the ETD Editor by clicking in the e*Gate Schema Designer tool bar.

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

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

Figure 12 Build Event Type Definition Dialog

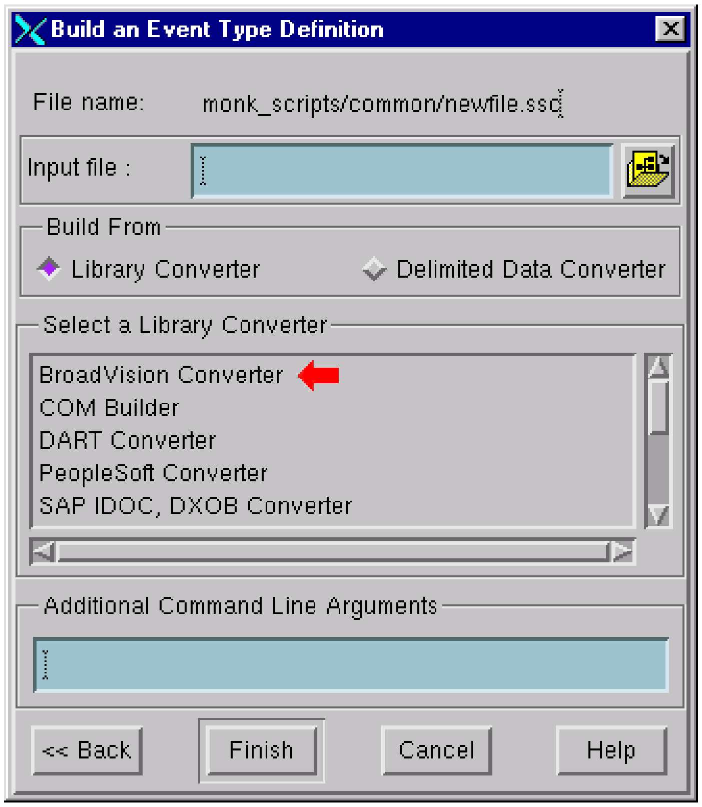


3 In the *File name* box, type the name of the ETD file you want to build.

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

4 Click **Next**. A new dialog box appears, as shown in Figure 13.

Figure 13 Building the ETD



Build an Event Type Definition

File name: monk_scripts/common/newfile.ssc

Input file :

Build From

[✓ Library Converter] Delimited Data Converter

Select a Library Converter

BroadVision Converter
COM Builder
DART Converter
PeopleSoft Converter
SAP IDOC, DXOB Converter

Additional Command Line Arguments

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

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

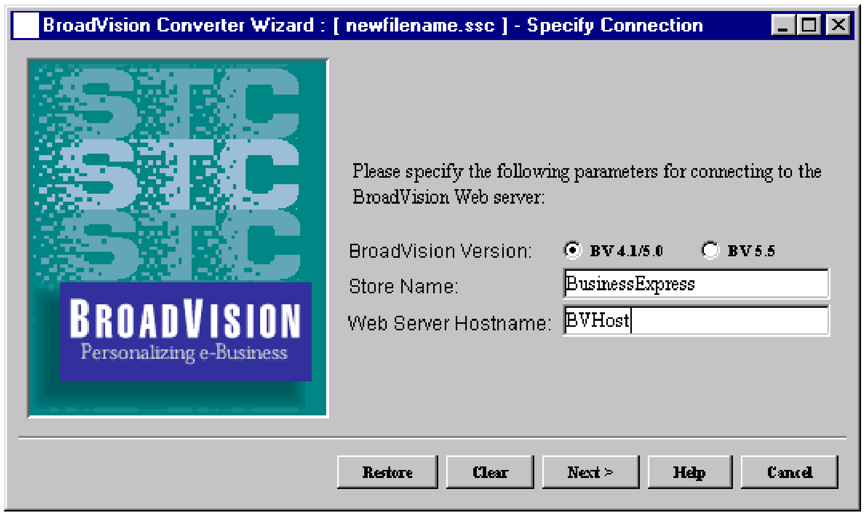
7 In the *Additional Command Line Arguments* box, type any additional arguments, if desired.

8 Click **Finish**, and the BroadVision Converter Wizard appears.

9 Follow the Wizard's instructions to finish building the ETD file.

3.4.2 The BroadVision Converter Wizard

Figure 14 BroadVision Converter Wizard (1)


BroadVision Converter Wizard : [newfilename.ssc] - Specify Connection
BROADVISION
Personalizing e-Business
Restore
Clear
Next >
Help
Cancel

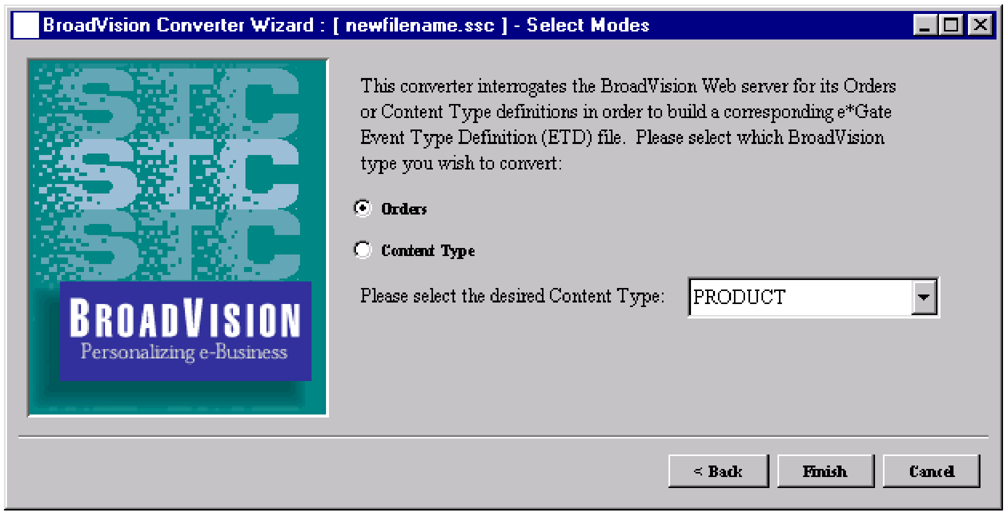
Please specify the following parameters for connecting to the BroadVision Web server:

BroadVision Version: [✓ BV 4.1/5.0] BV 5.5
Store Name: BusinessExpress
Web Server Hostname: BVHost

1 Select whether you are using BroadVision **4.1** or **5.0**, or BroadVision **5.5**.

2 Type in the **Store Name** and **Web Server Hostname**, and click **Next**.

Figure 15 BroadVision Converter Wizard (2)


BroadVision Converter Wizard : [newfilename.ssc] - Select Modes
BROADVISION
Personalizing e-Business
< Back
Finish
Cancel

This converter interrogates the BroadVision Web server for its Orders or Content Type definitions in order to build a corresponding e*Gate Event Type Definition (ETD) file. Please select which BroadVision type you wish to convert:

[✓ Orders]
Content Type

Please select the desired Content Type: PRODUCT

3 On the second Wizard screen, select which BroadVision type you want to convert, **Orders** or **Content Type**.

4 If you select **Content Type**, you also need to select the specific one from the scroll box.

5 Click **Finish**.

6 The Converter now runs and, when processing is completed, you are presented with the **Editor** screen.

3.4.3 Assigning ETDs to Event Types

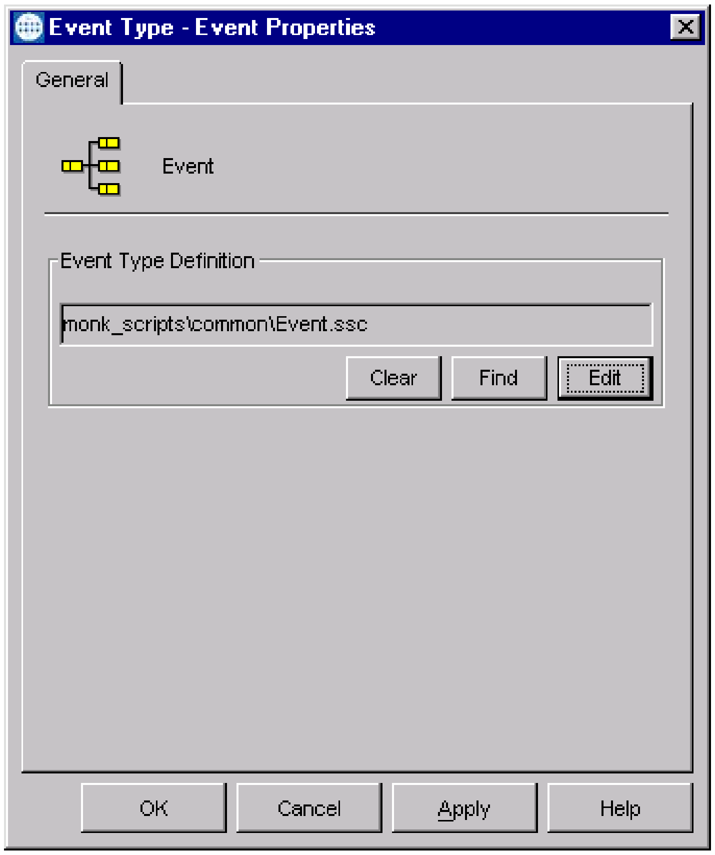
After you have created the e*Gate system's ETD files, you can assign them to Event Types you have already created.

To assign ETDs to Event Types

1. In the 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 16.

Figure 16 Event Type Properties Dialog Box



monk_scripts\common\Event.ssc

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

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

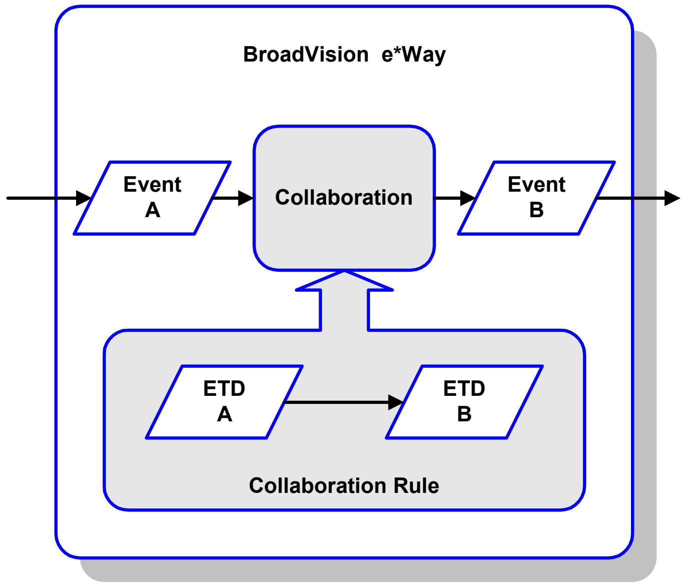
Each Event Type is now associated with the specified Event Type Definition.

3.5 Defining Collaborations

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

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

Figure 17 Collaborations



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

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

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

- **BV_Orders_Post** - example for Sales Orders data (only)
- **BV_Products** - example for Product data (only)

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

See **Optional Example Files** on page 24 for installation instructions.

3.7.1 BV_Orders_Post

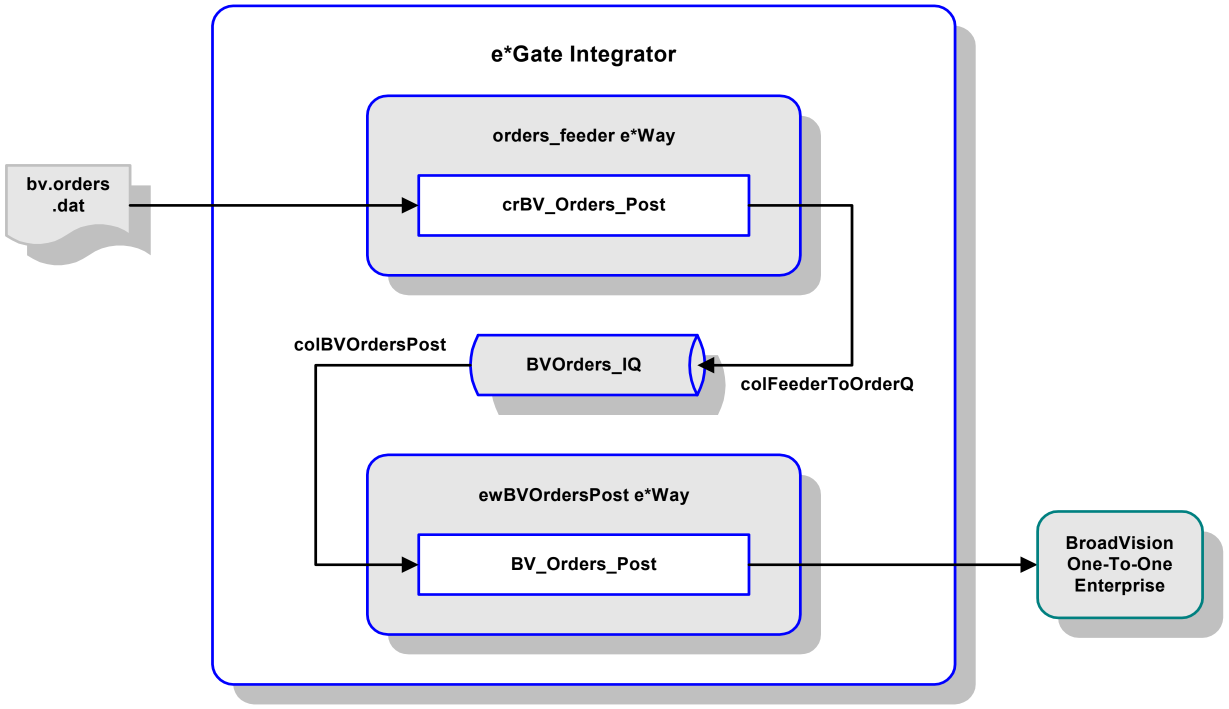
The e*Gate-to-BroadVision example, **BV_Orders_Post**, sets up a single instance of the BroadVision e*Way and also of the File e*Way, having the logical names shown in the following table.

e*Way Type	Logical Name
BroadVision e*Way	ewBVOrdersPost
File e*Way	orders_feeder

It also sets up an Intelligent Queue, with the logical name **BVOrders_IQ**.

The process within e*Gate Integrator is diagrammed in Figure 21.

Figure 18 BV_Products Schema



1 The File e*Way **orders_feeder** receives a data file, **bv.orders.dat**, from an external source.

2 Using the Pass-Through Collaboration **crBV_Orders_Post**, the e*Way **orders_feeder** then publishes the data to the IQ as **colFeederToOrderQ**.

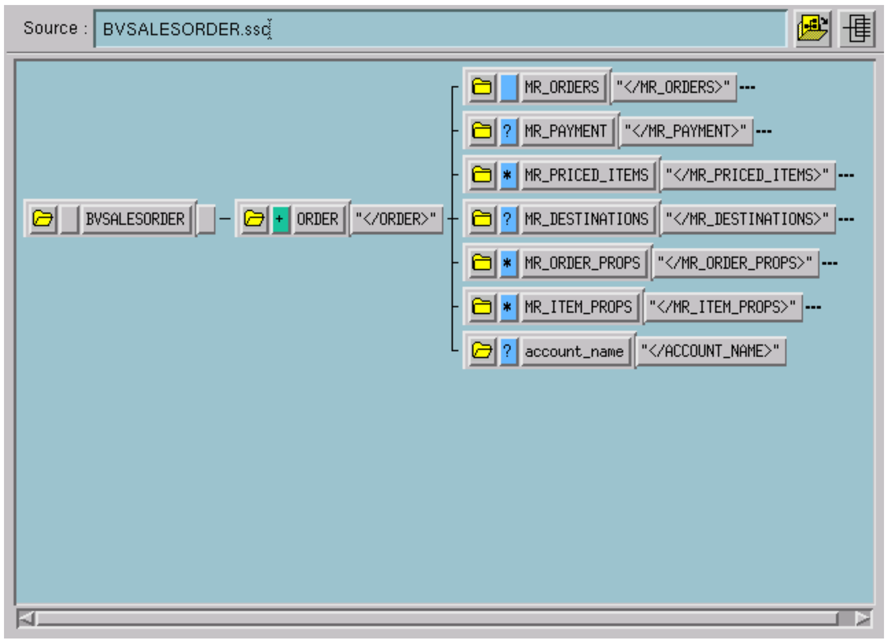
3 The BroadVision e*Way **ewBVOrdersPost** subscribes to the data from the IQ as **colBVOrdersPost**.

4 Using the Collaboration **BV_Orders_Post**, the e*Way **ewBVOrdersPost** transforms and sends the data to BroadVision in the required format.

Collaboration: BV_Orders_Post

This Collaboration is based on the Monk script **BV_Orders_Post.tsc**, whose source and destination ETDs are both **BVORDERSPOST.ssc** (see Figure 19 and Figure 20).

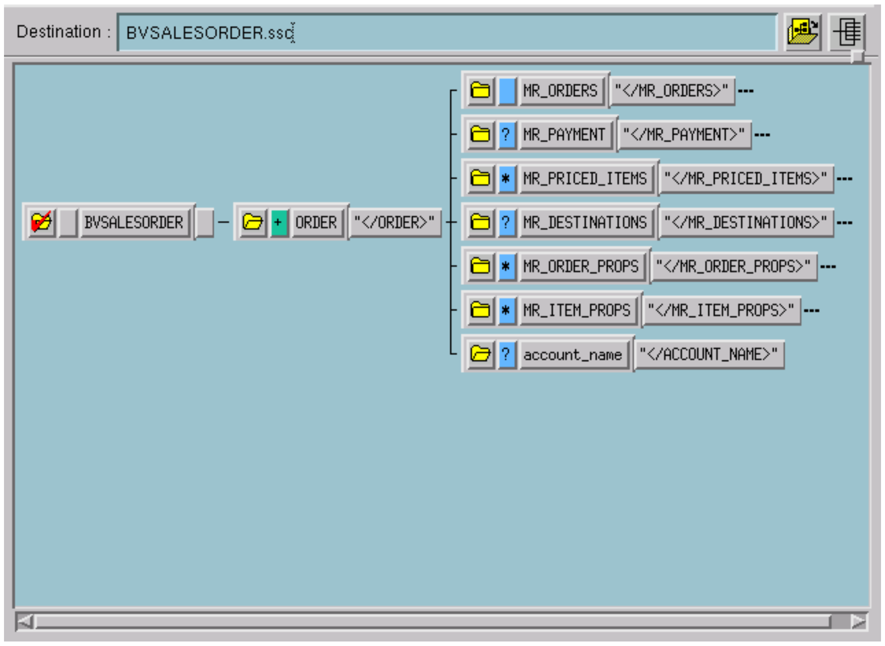
Figure 19 Source ETD



Source : BVSALESORDER.ssc

BVSALESORDER — + ORDER "</ORDER>"
- MR_ORDERS "</MR_ORDERS>"
- ? MR_PAYMENT "</MR_PAYMENT>"
- * MR_PRICED_ITEMS "</MR_PRICED_ITEMS>"
- ? MR_DESTINATIONS "</MR_DESTINATIONS>"
- * MR_ORDER_PROPS "</MR_ORDER_PROPS>"
- * MR_ITEM_PROPS "</MR_ITEM_PROPS>"
- ? account_name "</ACCOUNT_NAME>"

Figure 20 Destination ETD



Destination : BVSALESORDER.ssc

BVSALESORDER — + ORDER "</ORDER>"
- MR_ORDERS "</MR_ORDERS>"
- ? MR_PAYMENT "</MR_PAYMENT>"
- * MR_PRICED_ITEMS "</MR_PRICED_ITEMS>"
- ? MR_DESTINATIONS "</MR_DESTINATIONS>"
- * MR_ORDER_PROPS "</MR_ORDER_PROPS>"
- * MR_ITEM_PROPS "</MR_ITEM_PROPS>"
- ? account_name "</ACCOUNT_NAME>"

3.7.2 BV_Products

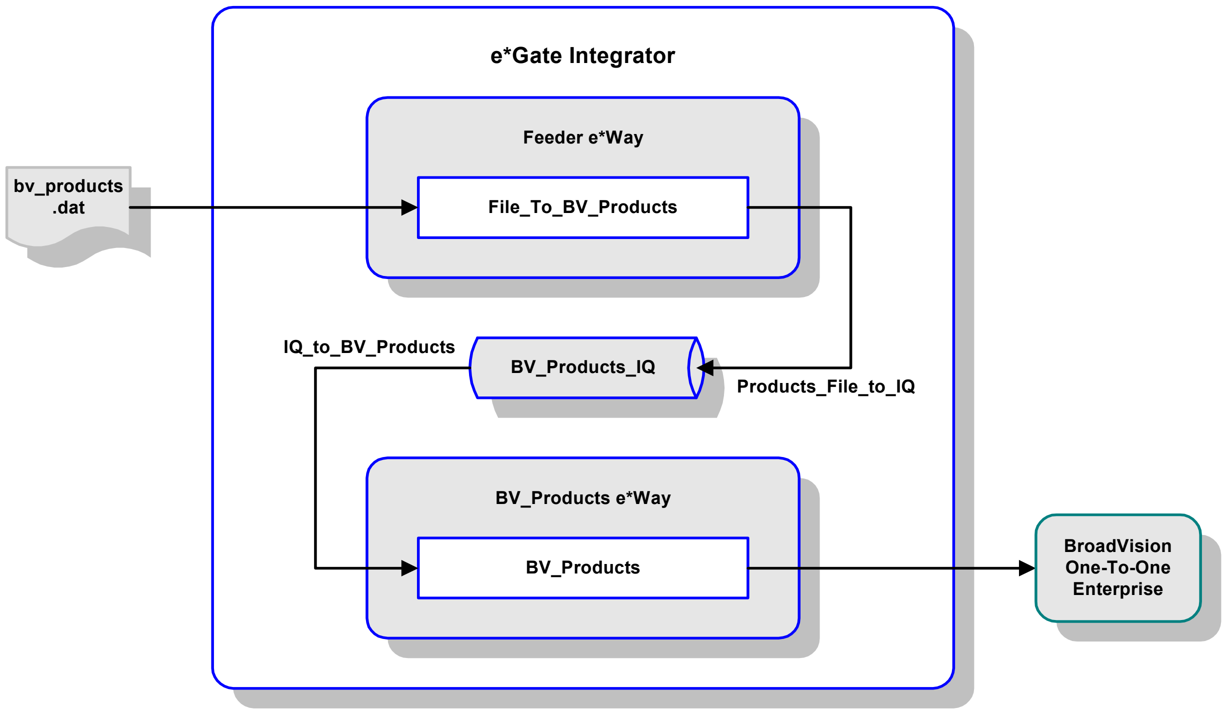
The e*Gate-to-BroadVision example, **BV_Products**, sets up a single instance of the BroadVision e*Way and also of the File e*Way, having the logical names shown in the following table.

e*Way Type	Logical Name
BroadVision e*Way	BV_Products
File e*Way	Feeder

It also sets up an Intelligent Queue, with the logical name **BV_Products_IQ**.

The process within e*Gate Integrator is diagrammed in Figure 21.

Figure 21 BV_Products Schema



1 The File e*Way **Feeder** receives a data file, **bv_products.dat**, from an external source.

2 Using the Pass-Through Collaboration **File_To_BV_Products**, the e*Way **Feeder** then publishes the data to the IQ as **Products_File_To_IQ**.

3 The BroadVision e*Way **BV_Products** subscribes to the data from the IQ as **IQ_To_BV_Products**.

4 Using the Collaboration **BV_Products**, the e*Way **BV_Products** transforms and sends the data to BroadVision in the required format.

Collaboration: BV_Products

This Collaboration is based on the Monk script **BV_Products.tsc**, whose source ETD is **BV_Products_Input.ssc** (see Figure 22) and destination ETD is **BV_Products.ssc** (see Figure 23).

Figure 22 Source ETD

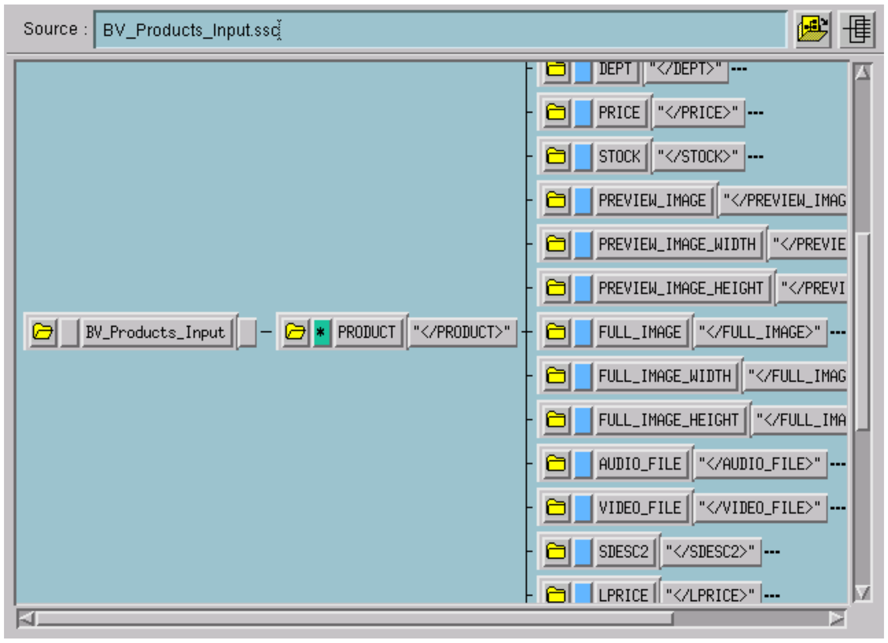
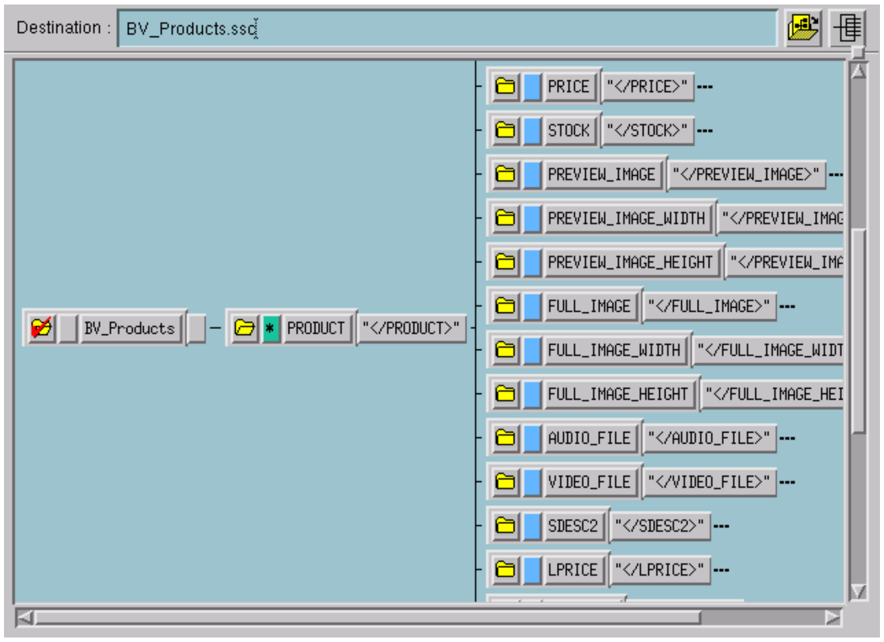

Source : BV_Products_Input.ssc

Figure 23 Destination ETD


Destination : BV_Products.ssc

Chapter 4

Setup Procedures

This chapter describes the procedure for customizing the BroadVision e*Way to operate with your system.

4.1 Overview

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

The topics discussed in this chapter include the following:

4.2 Setting Up the e*Way

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

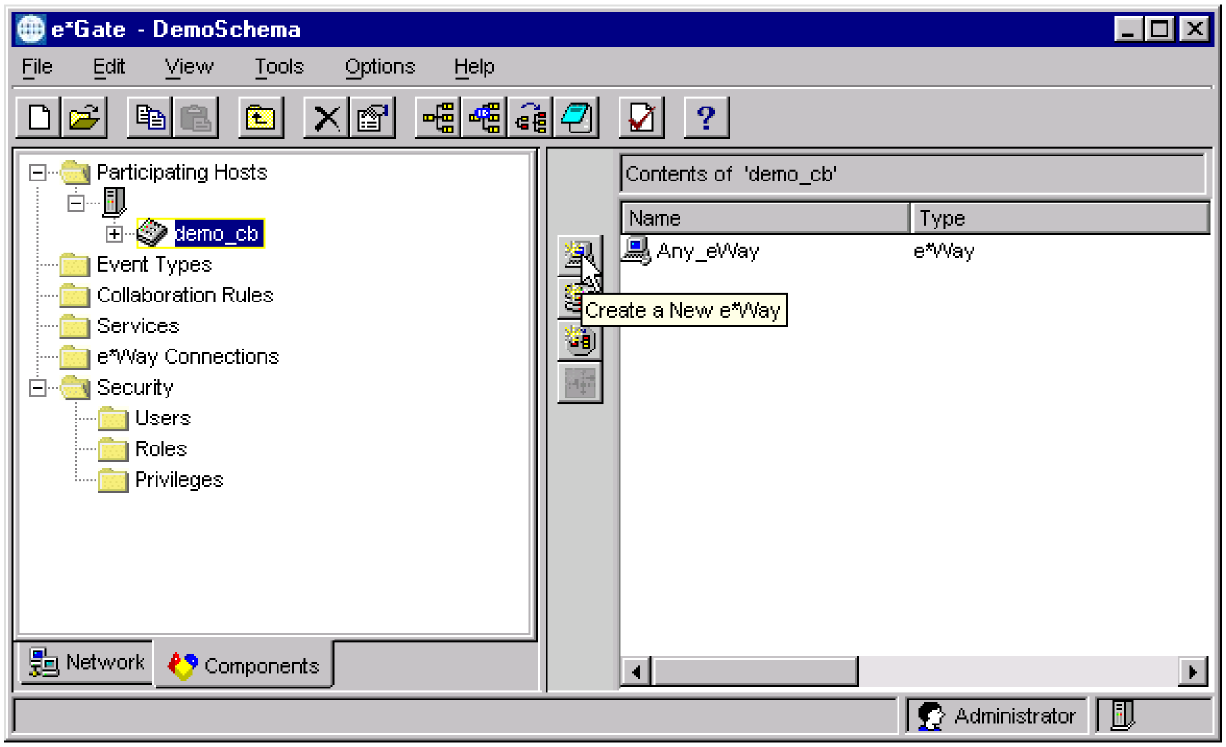
4.2.1 Creating the e*Way

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

To create an e*Way

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

Figure 24 e*Gate Schema Designer Window (Components View)



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

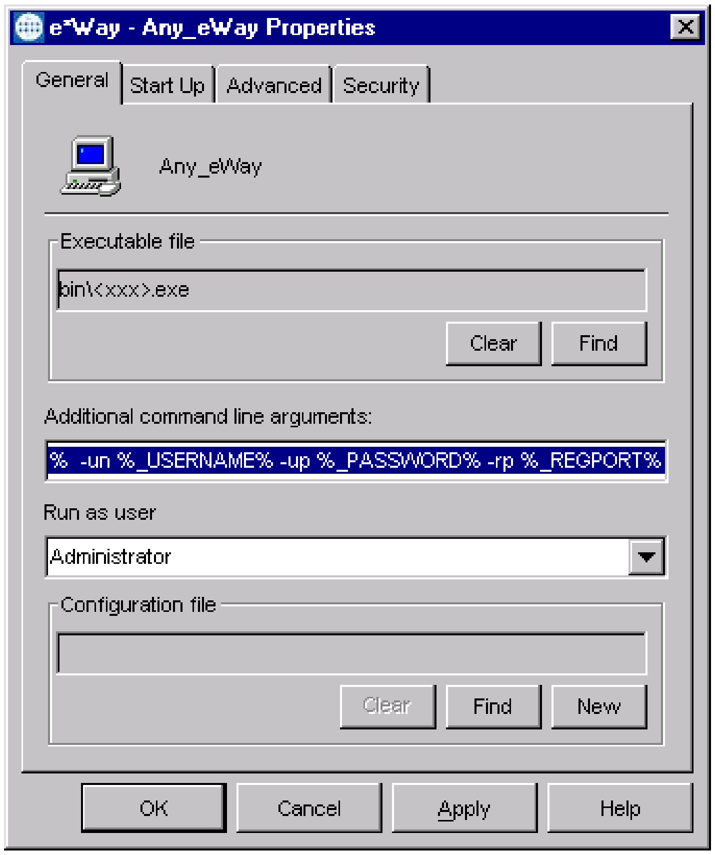
4.2.2 Modifying e*Way Properties

To modify any e*Way properties

1 Right-click on the desired e*Way and select **Properties** to edit the e*Way's properties. The properties dialog opens to the **General** tab (shown in Figure 25).

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

Figure 25 e*Way Properties (General Tab)



Executable file: bin\<xxx>.exe

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

Run as user: Administrator

Configuration file:

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

4.2.3 Configuring the e*Way

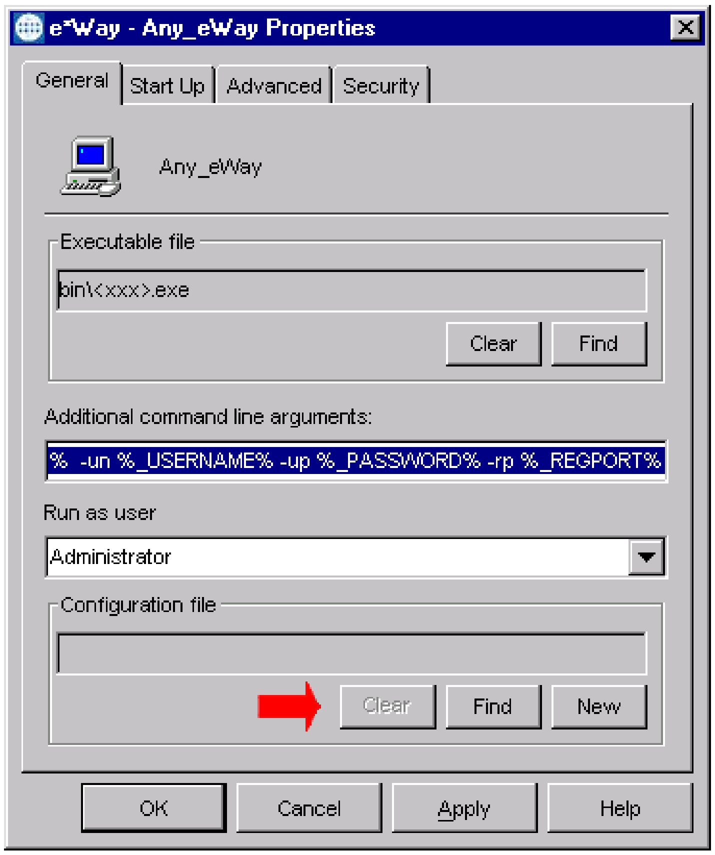
The e*Way's default configuration parameters are stored in an ASCII text file with a **.def** extension. The e*Way Editor provides a simple graphical interface for viewing and changing those parameters to create a working configuration (**.cfg**) file.

To change e*Way configuration parameters

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

Note: The default configuration file is **ewbv.def**.

Figure 26 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

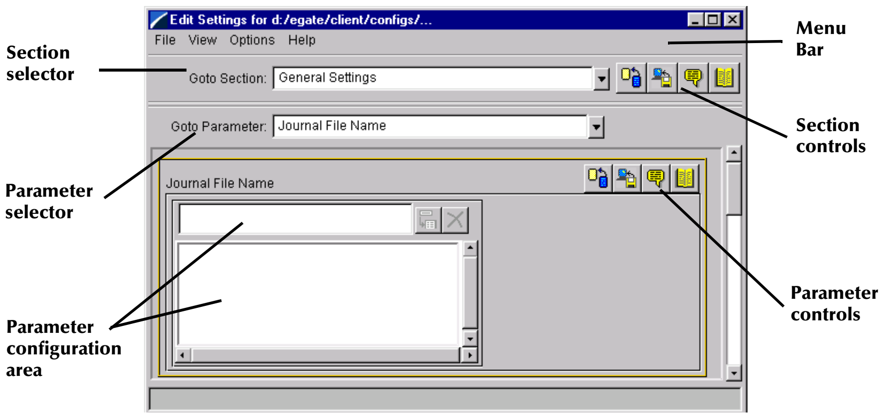
Configuration file:

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.

4.2.4 Using the e*Way Editor

Figure 27 The e*Way Configuration Editor


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

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

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

Section and Parameter Controls

The section and parameter controls are shown in Table 10 below.

Table 10 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 11

Table 11 Selection List Controls

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

Command-line Configuration

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

Getting Help

To launch the e*Way Editor's Help system

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

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

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

To display tips regarding the selected Configuration Section

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

To display tips regarding the selected Configuration Parameter

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

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

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

4.2.5 Changing the User Name

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

To change the user name

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

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

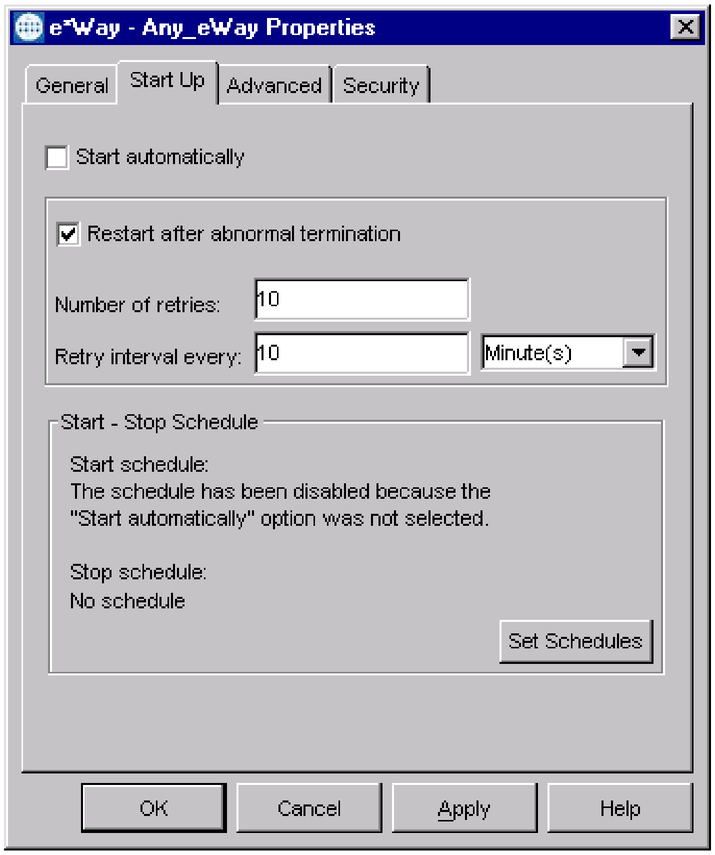
4.2.6 Setting Startup Options or Schedules

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

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

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see Figure 28). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 28 e*Way Properties (Start-Up Tab)



[ ] Start automatically

[✓] Restart after abnormal termination

Number of retries: 10

Retry interval every: 10 Minute(s)

Start - Stop Schedule

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

Stop schedule:
No schedule

To set the e*Way's startup properties

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

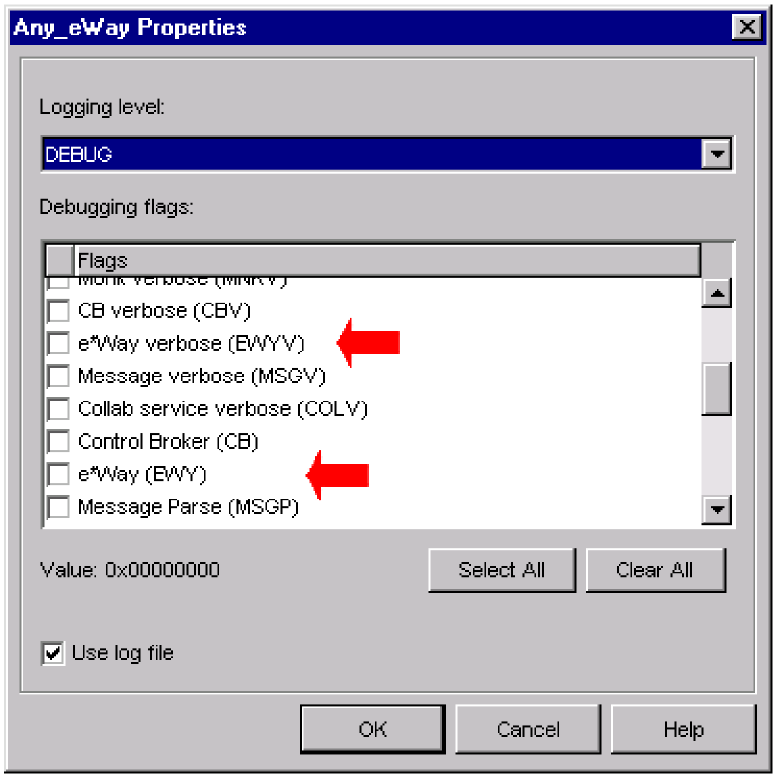
4.2.7 Activating or Modifying Logging Options

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

To set the e*Way debug level and flag

1. Display the e*Way's Properties dialog.
2. Select the **Advanced** tab.
3. Click **Log**. The dialog window appears (see Figure 29).

Figure 29 e*Way Properties (Advanced Tab - Log Option)



Any_eWay Properties

Logging level: DEBUG

Debugging flags:

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

Value: 0x00000000

- [x] Use log file

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

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

4.2.8 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way sends a Monitoring Event to the Control Broker, which routes it to the 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 Starting and Running the e*Way

The BroadVision e*Way must be run on the same machine as BroadVision One-To-One Enterprise (or Business Commerce). You should start the BroadVision One-To-One application *before* starting the e*Way, so that the e*Way can bind to the BroadVision One-To-One services. Note that BroadVision One-To-One Enterprise and Business Commerce are started manually, and the BroadVision starting process automatically appends the directories **bv1to1\bin** and **bv1to1\orbix\bin** to the Path environment variable.

4.3.1 Starting the e*Way Manually

The BroadVision One-To-One Enterprise or Business Commerce for Windows 2000 installation includes the **MKS Kit**, which is a Windows-based Korn shell application. Both the e*Way and the Control Broker can be launched manually from the Korn shell window, and the environment can be accessed by typing **env**. The Korn shell window can be launched from the **Start** > **Program** menu by running the file **bv1to1.sh** in the BroadVision One-To-One application folder.

4.4 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.

4.4.1 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 the *path* environment variable includes the location of the BroadVision dynamically-loaded libraries. The name of this variable on the different operating systems is:
  - PATH (Windows)
  - LD_LIBRARY_PATH (Solaris)
  - SHLIB_PATH (HP-UX)

In the BroadVision Application

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

4.4.2 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 Schema Monitoring system to monitor operations and performance.

Chapter 5

Operational Overview

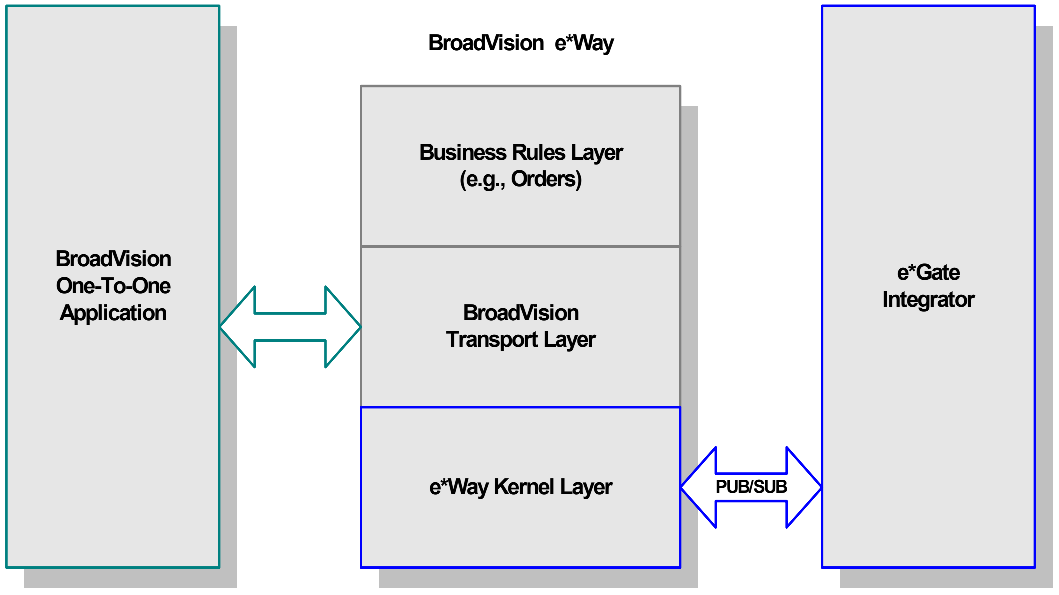
This chapter contains an overview of the architecture and basic internal processes of the BroadVision e*Way.

5.1 BroadVision e*Way Architecture

Note: *This section describes functionality that is common to all e*Ways based on the Generic e*Way Kernel. Not all of this common functionality is used routinely by the BroadVision e*Way.*

Conceptually, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers that handle communication with the external application, built upon an e*Way Kernel layer that manages the processing of data and subscribing or publishing to other e*Gate components (see Figure 30).

Figure 30 BroadVision e*Way Architecture


BroadVision e*Way
Business Rules Layer (e.g., Orders)
BroadVision One-To-One Application
BroadVision Transport Layer
e*Gate Integrator
e*Way Kernel Layer
PUB/SUB

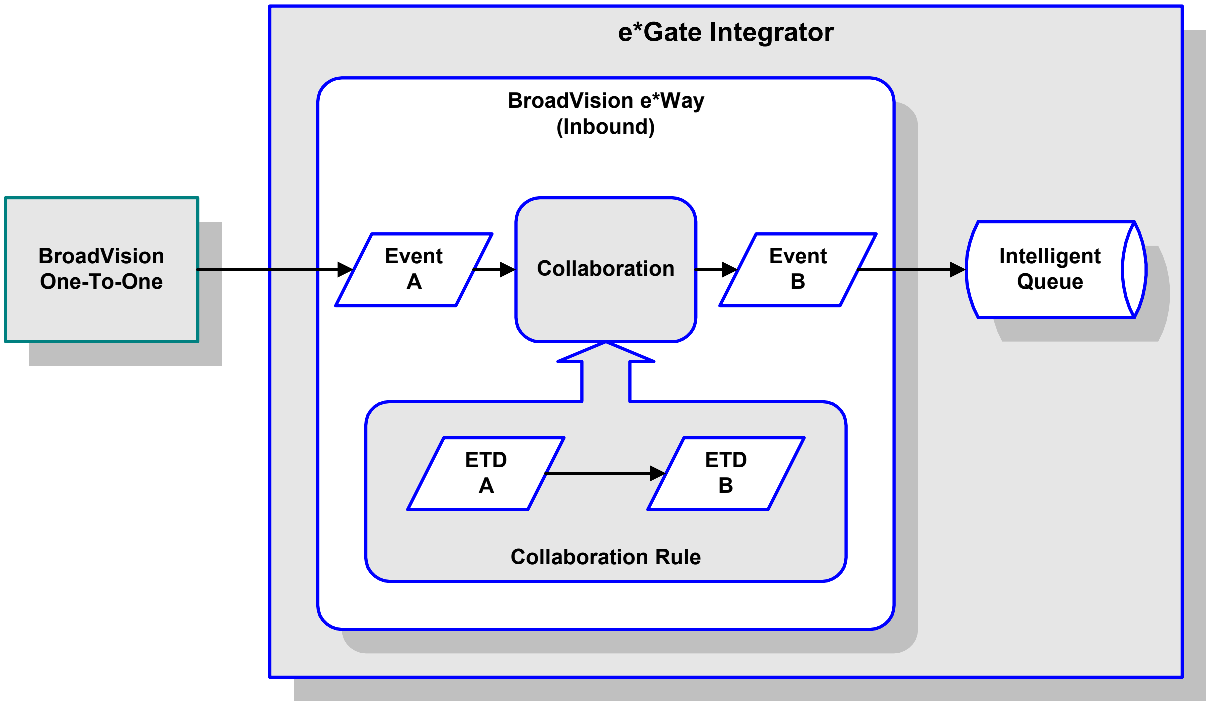
The upper layers of the e*Way use Monk functions to perform Business Process modeling and ETD mapping, package data as e*Gate *Events*, send those Events to

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

Figure 31 Collaborations



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

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

5.2 Basic e*Way Processes

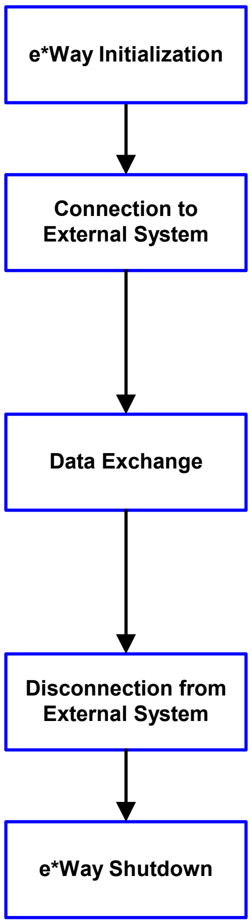
Note: *This section describes the basic operation of a typical e*Way based on the Generic e*Way Kernel. Not all functionality described in this section is used routinely by this e*Way.*

The most basic processes carried out by an e*Way are listed in Figure 32. 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 32 Basic e*Way Processes

Process

Monk Configuration Sections


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

Startup Function on page 70 (also see **Monk Environment Initialization File** on page 69)

External Connection Establishment Function on page 72
External Connection Verification Function on page 73

Event-driven Data Exchange
Process Outgoing Message Function on page 70

Schedule-driven Data Exchange
Exchange Data with External Function on page 71
Process Outgoing Message Function on page 70
Negative Acknowledgment Function on page 74

External Connection Shutdown Function on page 73

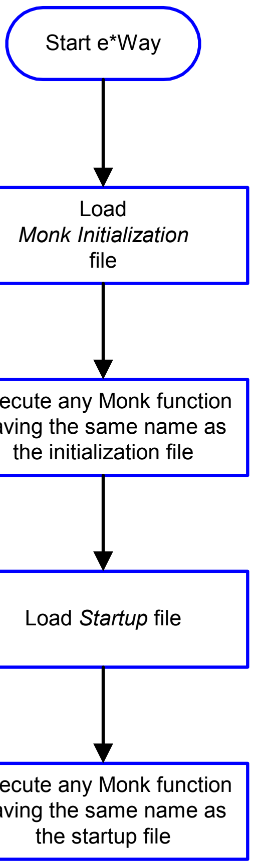
Shutdown Command Notification Function on page 75

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

Initialization Process

Figure 33 illustrates the e*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

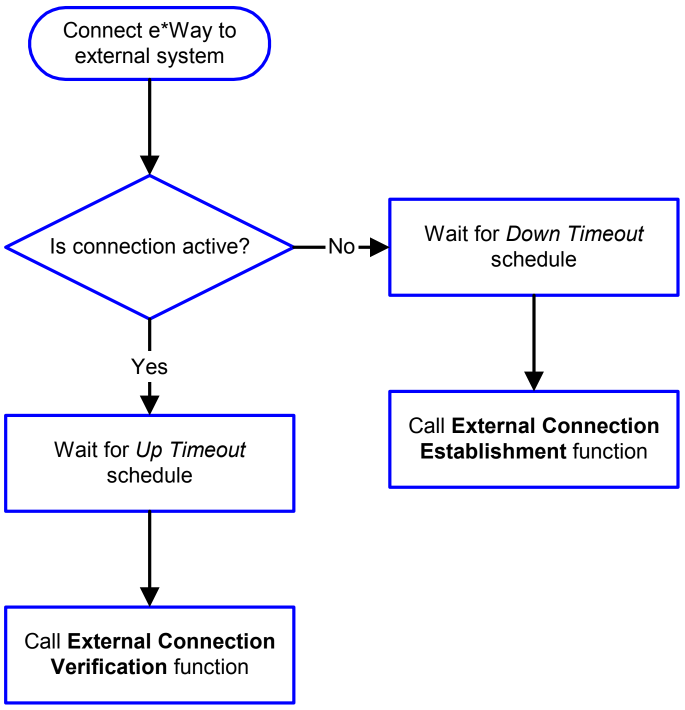
Figure 33 Initialization Process


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

Figure 34 Connection Process


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

Note: *The e*Way selects the connection function based on an internal **up/down** flag rather than a poll to the external system. See* **Figure 36 on page 60** *and* **Figure 35 on page 59** *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 106 and* **send-external-down** *on page 106 for more information.*

Data Exchange Process

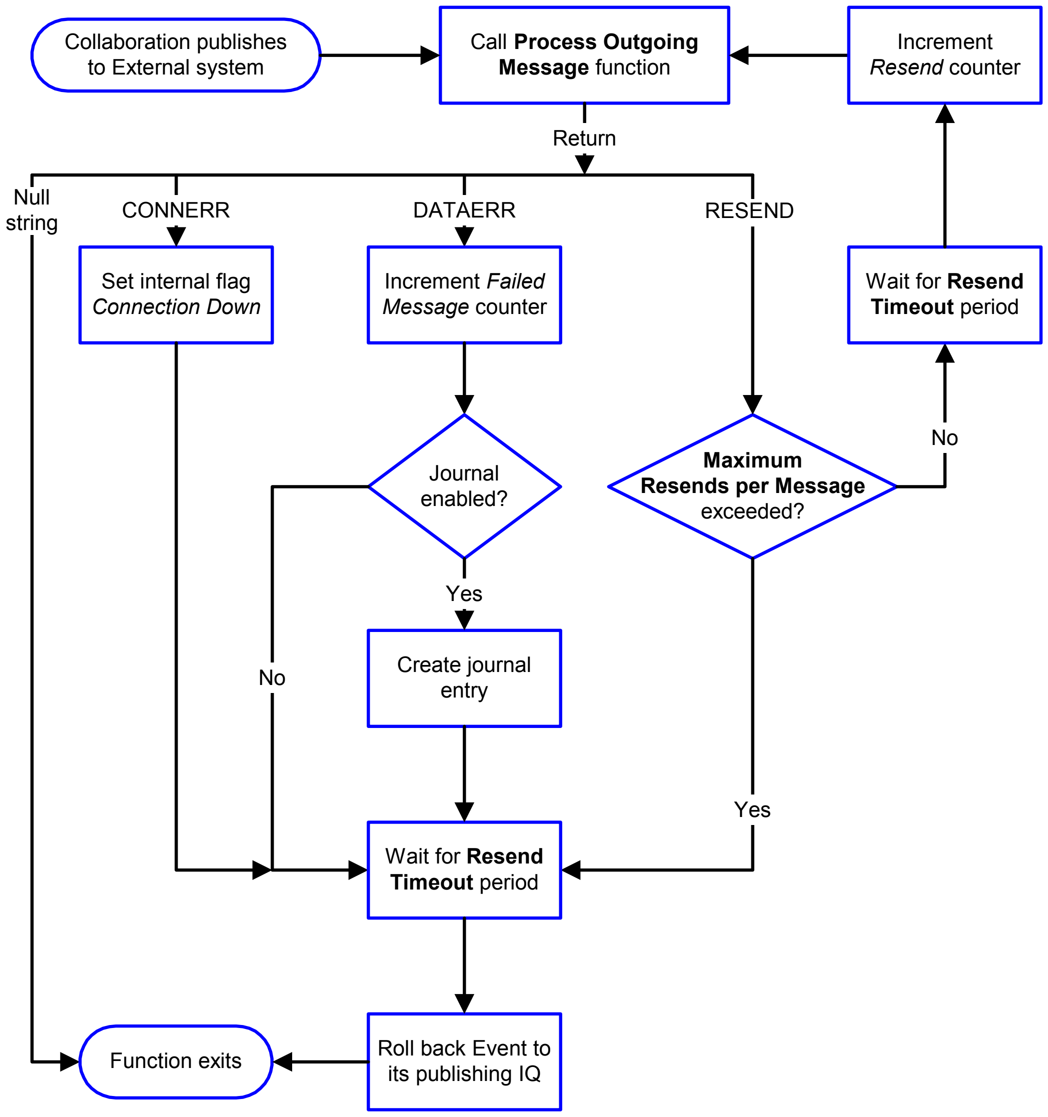
Event-driven

Figure 35 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.

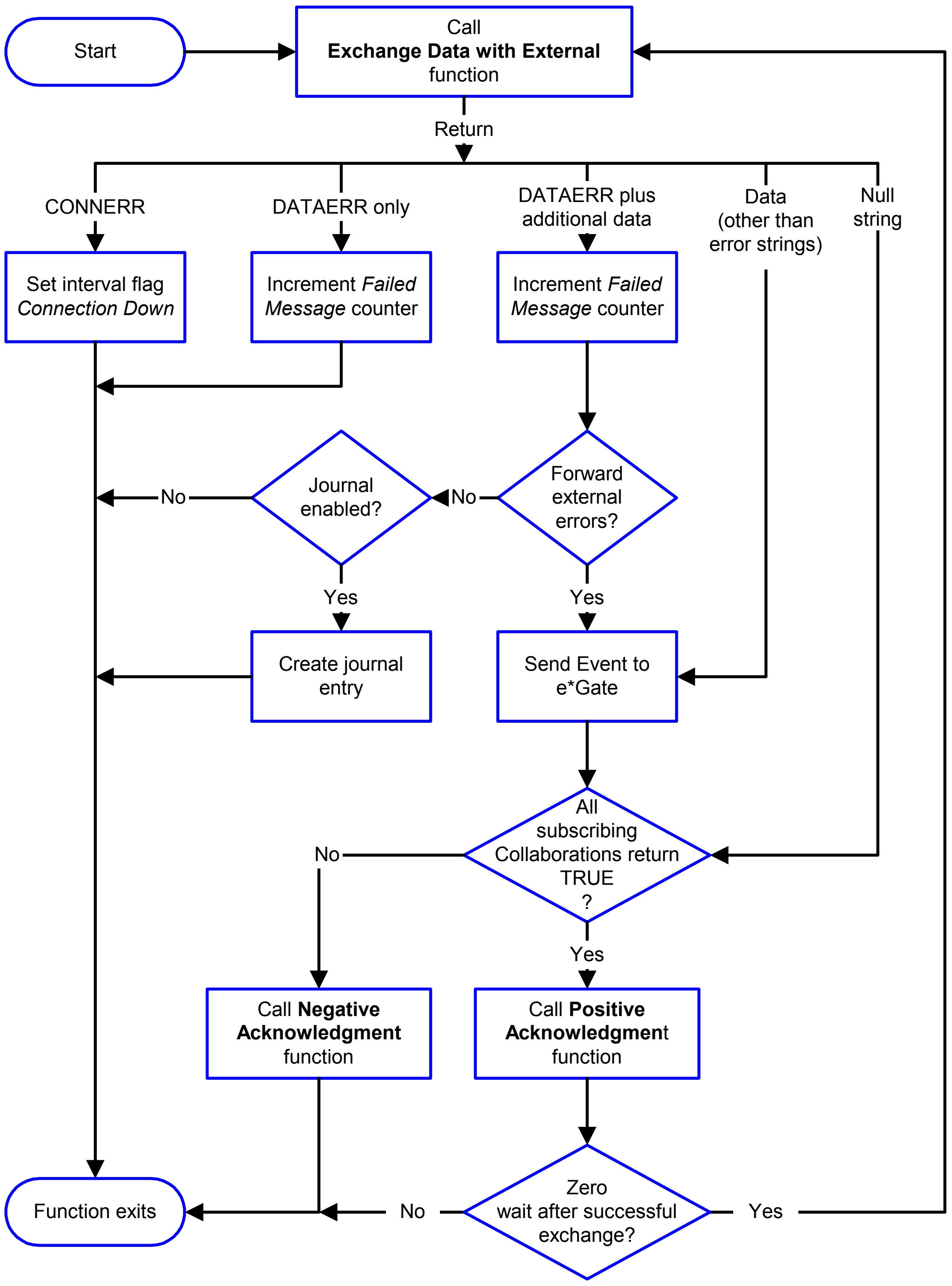
Figure 35 Event-Driven Data Exchange Process


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

Schedule-driven

Figure 36 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 36 Schedule-Driven Data Exchange Process


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

Start can occur in any of the following ways:

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

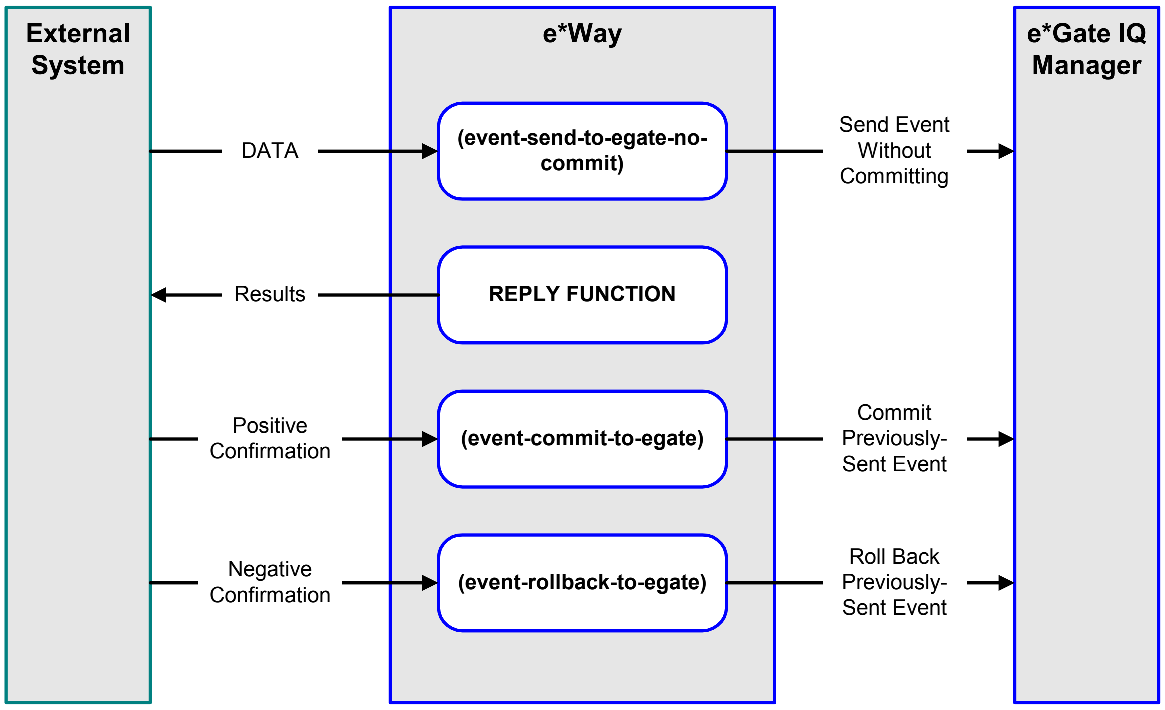
*Send Events to e*Gate* can be implemented using any of the following Monk functions:

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

The last of these is used when confirmation of correct transmission is required from the external system. In this case, the e*Way sends information back to the external system after receiving data. Depending upon whether the acknowledgment is positive or negative, you subsequently use one of the following functions to complete the process (see Figure 37):

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

Figure 37 Send Event to e*Gate with Confirmation

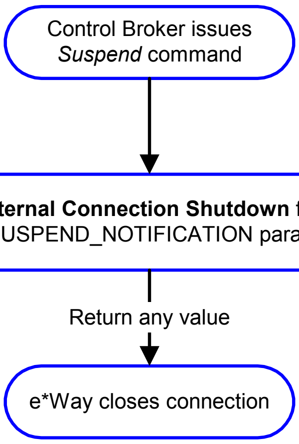

External System
e*Way
e*Gate IQ Manager
DATA
(event-send-to-egate-no-commit)
Send Event Without Committing
Results
REPLY FUNCTION
Positive Confirmation
(event-commit-to-egate)
Commit Previously-Sent Event
Negative Confirmation
(event-rollback-to-egate)
Roll Back Previously-Sent Event

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

Disconnect from External Process

Figure 38 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

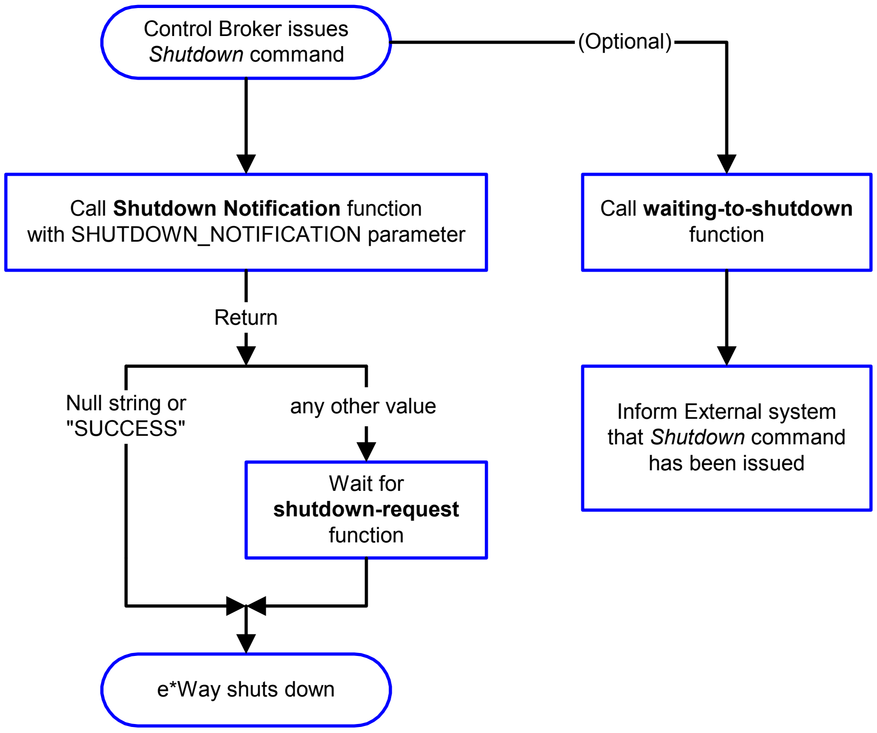
Figure 38 Disconnect Process


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

Shutdown Process

Figure 39 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

Figure 39 Shutdown Process


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

Chapter 6

Configuration Parameters

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

6.1 Overview

The e*Way's configuration parameters are set using the e*Way Editor; see **Configuring the e*Way** on page 43 for procedural information. The BroadVision e*Way's configuration parameters are organized into the following sections. The default configuration is provided in **ewbv.def**.

General Settings on page 64

Communication Setup on page 66

Monk Configuration on page 68

BroadVision Settings on page 77

6.2 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

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

Additional Information

An Event is Journaled for the following conditions:

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

Max Resends Per Message

Description

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

Required Values

An integer between **1** and **1,024**. The default is **5**.

Max Failed Messages

Description

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

Forward External Errors

Description

Selects whether or not error messages received from the external system that begin with the string **"DATAERR"** is queued to the e*Way's configured queue. See **Exchange Data with External Function** on page 71 for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages are not to be forwarded. See **Data Exchange Process** on page 59 for more information about how the e*Way uses this function.

6.3 Communication Setup

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

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

Start Exchange Data Schedule

Description

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

Required Values

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

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

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

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

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).

Exchange Data Interval

Description

Determines the number of seconds the e*Way waits between Event exchange attempts.

Required Values

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

Down Timeout

Description

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

Required Values

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

Up Timeout

Description

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

Required Values

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

Resend Timeout

Description

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

Required Values

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

Zero Wait Between Successful Exchanges

Description

Specifies whether or not to initiate data exchange after the **Exchange Data Interval**, or immediately after a successful previous exchange.

Required Values

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

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

If this parameter is set to **No**, the e*Way always waits the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External Function**.

6.4 Monk Configuration

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

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 **ewbv-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 **bv-startup**.

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

Returns

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

Process Outgoing Message Function

Description

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

Required Values

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

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

Returns

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

Additional Information

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

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

Exchange Data with External Function

Description

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

Required Values

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

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

Returns

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

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

Additional Information

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

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

External Connection Establishment Function

Description

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

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

Required Values

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

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

Returns

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

External Connection Verification Function

Description

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

Required Values

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

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

Returns

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

Additional Information

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 **ewbv-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 **bv-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 **bv-nak**.

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

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. There is no default value for this parameter.

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 BroadVision Settings

These configuration parameters set up the BroadVision application.

Version

Description

The version of BroadVision One-To-One Enterprise being used.

Required Values

Either of the following:

- BV4.1/5.0 (default)
- BV5.5

Store Name

Description

Name of store or business.

Required Values

String. Default: (none).

Agent Name

Description

The Agent name for this e*Way for the purpose of accessing the BroadVision system.

Required Values

String. Default: (none).

Desired State

Description

Desired state for order retrieval from BroadVision (only orders in the desired state are retrieved by the e*Way).

Required Values

Number between **0** and **999** (see **Predefined States** on page 79).

Default: **1**

New State

Description

After orders in the desired state are read from BroadVision and sent to the IQ, the order state is updated to the new state.

Required Values

Number between **0** and **999** (see **Predefined States** on page 79).

Default: **3**

Maximum Order Count

Description

The maximum number of orders to get with a single poll to the BroadVision system.

Required Values

Range: **1-84,600**

Default: **1**

Maximum Content Count

Description

The maximum number of content references to get from the BroadVision system.

Required Values

Range: **1-84,600**

Default: **1**

Content Status

Description

Specifies whether to list *all* contents, or only *online* contents from the BroadVision system.

Required Values

Either **ALL** or **ONLINE**; the default is **ALL**.

Predefined States

Number	Name	Message String
0	UnknownOrderState	“Unknown Order State”
1	OrderNew	“New Order”
2	BeingAuthorized	“Payment is being Authorized”
3	WaitFulfillment	“Order is waiting to be Fulfilled”
4	BeingFulfilled	“Order is being Fulfilled”
5	OrderFulfilled	“Order is completely fulfilled”
6	OrderPartiallyFulfilled	“Order is partially fulfilled”
7	WaitReturn	“Payment is waiting to be return/credit back”
8	BeingFullySettled	“Payment is being fully settled”
9	BeingPartiallySettled	“Payment is being partially settled”
10	BeingReturnSettled	“Payment is being return settled”
11	OrderComplete	“Order is complete (fully settled)”
12	OrderPartiallyComplete	“Order is partially complete/settled”
13	OrderReturnComplete	“Payment return/credit complete successfully”
14	OrderCancelled	“Order is cancelled”
101	AuthorizationError	“Cannot authorize payment”
102	FulfillmentError	“Cannot fulfill order”
103	SettlementError	“Cannot settle order”
104	OrderReturnError	“Payment return/credit back failed”
999	UnknownError	“Unknown Error”

Chapter 7

API Functions

7.1 Overview

As stated earlier, the e*Way can be viewed as a three-layered structure, consisting of a:

- Business Rules Layer
- BroadVision Transport Layer
- e*Way Kernel Layer

Each layer contains Monk scripts and/or functions, and makes use of lower-level Monk functions residing in the layer beneath. Developers primarily make use of the functions residing in the upper (BroadVision) layers.

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

- **BroadVision Orders Functions** on page 81
- **BroadVision General Functions** on page 88
- **Generic e*Way Functions** on page 102

7.2 BroadVision Orders Functions

These Monk functions have been developed specifically to support the **Orders** business process of a BroadVision One-To-One application.

bv-order-complete-fulfill

Description

Used to *completely* fill the specified order.

Signature

```
(bv-order-complete-fulfill <order_number_string>)
```

Parameters

Name	Type	Description
order_number_string	String	BroadVision order number.

Returns

Boolean true (#t) upon success; otherwise, false (#f).

Throws

None

Location

bv-order-complete-fulfill.monk

bv-order-get-accountname

Description

Used to get the **account_name** that corresponds to the given **user_id**.

Signature

```
(bv-order-get-accountname <user_id_number>)
```

Parameters

Name	Type	Description
user_id_number	Num	User ID.

Returns

Upon success, a string representing the account name that corresponds to the given user ID.

Upon failure, a Boolean false (#f).

Throws

None

Location

bv-order-get-accountname.monk

bv-order-get-ordernumber

Description

Looks up the BroadVision order number based on the SAP order number property.

Signature

```
(bv-order-get-ordernumber <sap_order_number_string>)
```

Parameters

Name	Type	Description
sap_order_number_string	String	SAP order number.

Returns

Upon success, a string containing the BroadVision order number.

Upon failure, a Boolean false (#f).

Throws

None

Location

bv-order-get-ordernumber.monk

bv-order-get-orderprop-ordernumber

Description

Looks up the BroadVision order number based on the property name and property value.

Signature

```
(bv-order-get-orderprop-ordernumber <prop_name>_<prop_value>)
```

Parameters

Name	Type	Description
prop_name	String	Corresponds to **PROP_NAME** in the **MR_ORDER_PROPS** table.
prop_value	String	Corresponds to **PROP_VALUE** in the **MR_ORDER_PROPS** table.

Returns

Upon success, a string containing the BroadVision order number.

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

Throws

None

Location

bv-order-get-orderprop-ordernumber.monk

Note: *The property name-value combination must be* ***unique*** *to obtain the correct BV order number.*

bv-order-get-orders

Description

Gets orders corresponding to the desired state. Both the state and the maximum number of orders to get are configured through the GUI.

Signature

```
(bv-order-get-orders)
```

Parameters

None.

Returns

Upon success, a string containing the contents of order (empty string is returned when there is no order of the desired state).

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

Throws

None

Location

bv-order-get-orders.monk

bv-order-get-useralias

Description

Used to get the user aliases for the given **user_id**.

Signature

```
(bv-order-get-useralias <user_id_number>)
```

Parameters

Name	Type	Description
user_id_number	Integer	User ID.

Returns

Upon success, returns a string containing user aliases for the given user_id.

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

Throws

None

Location

bv-order-get-useralias.monk

bv-order-get-userid

Description

Used to get a list of user IDs for the given account.

Signature

```
(bv-order-get-userid <account_name_string>)
```

Parameters

Name	Type	Description
account_name_string	String	Account name.

Returns

Upon success, a vector representing the user IDs associated with the account are returned in a vector of strings.

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

Throws

None

Location

bv-order-get-userid.monk

bv-order-partial-fulfill

Description

Used to *partially* fill the specified order.

Signature

```
(bv-order-partial-fulfill <order_number_string>)
```

Parameters

Name	Type	Description
order_number_string	String	BroadVision order number.

Returns

Boolean true (**#t**) upon success; otherwise, false (**#f**).

Throws

None

Location

bv-order-partial-fulfill.monk

bv-order-set-configured-state

Description

Sets a new state (as configured through the GUI) for the given order number.

Signature

```
(bv-order-set-configured-state <order_number_string>)
```

Parameters

Name	Type	Description
order_number_string	String	BroadVision order number.

Returns

Boolean true (**#t**) if the new state is set successfully; otherwise, false (**#f**).

Throws

None

Location

bv-order-set-state.monk

bv-order-start

Description

Prepares to create/update an order.

Signature

```
(bv-order-internal <ordernumber_string>)
```

Parameters

Name	Type	Description
ordernumber_string	String	

Returns

Boolean true (**#t**) upon success; otherwise, false (**#f**).

Throws

None

Location

bv-order-internal.monk

bv-order-struct-create

Description

Creates an order with the specified fields in **node-path**.

Signature

```
(bv-order-struct-create <node-path>)
```

Parameters

Name	Type	Description
node-path	Path	Path to node containing specified field.

Returns

Upon success, the order number for the new object.

Upon failure, a Boolean false (#f).

Throws

None

Location

bv-order-struct-create.monk

Note: *Only those fields whose* **MAP** *subfield has a value of* **1** *are used.*

bv-order-struct-update

Description

Modifies an order with the specified fields in **node-path**.

Signature

```
(bv-order-struct-update <node-path>)
```

Parameters

Name	Type	Description
node-path	Path	Path to node containing specified field.

Returns

Upon success, the order number for the new object.

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

Throws

None

Location

bv-order-struct-update.monk

Note: *Only those fields whose* **MAP** *subfield has a value of* **1** *are used.*

7.3 BroadVision General Functions

These Monk functions have been developed specifically to control communications between the BroadVision e*Way and the BroadVision One-To-One application, and are external to the e*Way kernel.

bv-startup

Description

A dummy **STARTUP** function.

Signature

```
(bv-startup)
```

Parameters

None.

Returns

The string **"SUCCESS"** indicates success, **"FAILURE"** indicates failure.

Throws

None

Location

bv.monk

bv-connect

Description

Initializes a connection to the BroadVision system; for example, initializing an order manager.

Signature

```
(bv-connect)
```

Parameters

None.

Returns

The string **"UP"** indicates connection is up (operational); **"DOWN"** indicates connection is down (non-operational).

Throws

None.

Location

bv.monk

bv-verify-connect

Description

Verifies the connection with the BroadVision system.

Signature

```
(bv-verify-connect)
```

Parameters

None.

Returns

The string **"UP"** indicates connection is up (operational); **"DOWN"** indicates connection is down (non-operational).

Throws

None

Location

bv.monk

---

bv-ack

Description

A dummy **ACK** function.

Signature

```
(bv-ack)
```

Parameters

None.

Returns

An empty string.

Throws

None

Location

bv.monk

---

bv-nak

Description

A dummy **NAK** function.

Signature

```
(bv-nak)
```

Parameters

None.

Returns

An empty string.

Throws

None

Location

bv.monk

bv-category-create

Description

Creates a new (child) category under the specified parent category, having the specified name.

Signature

```
(bv-category-create <cat_name> <parent_oid> <store_id> <content_type>
    <status>)
```

Parameters

Name	Type	Description
cat_name	String	Category name.
parent_oid	Integer	Object ID of parent category.
store_id	Integer	Store ID number.
content_type	Integer	Type of content.
status	Integer	Status.

Returns

Upon success, returns an integer representing the object ID.

Upon failure, a Boolean false (#f).

Throws

If you create a category that already exists, you get a Monk exception.

Location

bv-category-create.monk

Examples

```
(display (bv-category-create "Printers" 0 101 0 1)) (newline) -8012
```

bv-category-delete

Description

Deletes a category having an object ID of “categorical”.

Signature

```
(bv-category-delete <category_oid>)
```

Parameters

Name	Type	Description
category_oid	OID	Object ID of desired category.

Returns

Boolean true (**#t**) if the category deletes successfully; otherwise, false (**#f**).

Throws

None

Location

bv-category-delete.monk

Examples

```
(display (bv-category-delete -8048)) (newline)
```

bv-category-get-cat-entry

Description

Gets the category entry for the named category.

Signature

```
(bv-category-get-cat-entry <category_name>)
```

Parameters

Name	Type	Description
category-oid	OID	Object ID of desired category.

Returns

A vector of 8 items from **bv_category** table:

```
#(oid type parent_oid store_id content_type status deleted always_0)
```

If category doesn't exist, it returns a vector of all zeros:

```
#(0 0 0 0 0 0 0 0)
```

Throws

None

Location

bv-category-get-cat-entry.monk

Examples

```
(display (bv-category-get-cat-entry "Computers")) (newline) #(-8047 2
   0 101 0 1 0 0)


(display (vector-ref (bv-category-get-cat-entry "Computers") 0)) -
   8047
```

bv-category-move

Description

Moves the specified category to a new branch, giving it the new object ID.

Signature

```
(bv-category-move <old_oid> <new_oid>)
```

Parameters

Name	Type	Description
old_oid	Integer	Current object ID of desired category.
new_oid	Integer	New object ID for specified category.

Returns

Boolean true (**#t**) if the function concludes successfully; otherwise, false (**#f**).

Throws

None

Location

bv-category-move.monk

Examples

```
(display (bv-category-move -8043 -8047)) (newline)
```

Note: *Supported only by BroadVision release 5.5 and above.*

bv-category-rename

Description

Renames the category having the specified object ID with the specified name.

Signature

```
(bv-category-rename <category_oid> <new_name>)
```

Parameters

Name	Type	Description
category_oid	OID	Object ID of category to be renamed.
new_name	String	New name for category having object ID of **category_oid**.

Returns

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

Throws

None

Location

bv-category-rename.monk

Examples

```
(display (bv-category-rename -8047 "NewCategoryName")) (newline)
```

bv-cnt-delete

Description

Used to delete the content object indicated by the content key value.

Signature

```
(bv-cnt-delete <content_type_name_string> <cnt_key_value_string>)
```

Parameters

Name	Type	Description
content-type-name-string	String	Content type name.
cnt-key-value-string	String	Content key value.

Returns

Boolean true (#t) if the category deletes successfully; otherwise, false (#f).

Throws

None

Location

bv-cnt-delete.monk

bv-cnt-get-productname

Description

Used to get the name of the product whose **prod_id** is specified.

Signature

```
(bv-cnt-get-productname <prod_id_string>)
```

Parameters

Name	Type	Description
prod_id	String	Product ID.

Returns

Upon success, a string representing the name of the specified product.

Upon failure, a Boolean false (#f).

Throws

None

Location

bv-cnt-get-productname.monk

bv-cnt-sql-select

Description

Used to query the database with the following **SQL** statement:

```
SELECT <field_name> FROM <table_name> WHERE <condition>.
```

Signature

```
(bv-cnt-sql-select <field_name_string> <table_name_string>
    <condition_string>)
```

Parameters

Name	Type	Description
field_name	String	Name of data field.
table_name	String	Name of table in which the data field is located.
condition	String	Location of table.

Returns

Upon success, a vector of Monk strings. Elements of this vector are values of the specified field, returned as strings.

Upon failure, a Boolean false (#f).

Throws

None

Location

bv-cnt-sql-select.monk

bv-cnt-struct-create

Description

Creates a content-type object with the specified fields in **node-path**. Note that the **user_id** defaults to **0**, as recommended by BroadVision.

Signature

```
(bv-cnt-struct-create <node-path>)
```

Parameters

Name	Type	Description
node-path	String	Path to node containing specified field.

Returns

Upon success, an integer representing the object ID number for the newly created object.

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

Throws

None

Location

bv-cnt-struct-create.monk

Note: *Only those fields whose* **MAP** *subfield has a value of* **1** *are used.*

bv-cnt-struct-update

Description

Updates the fields of a content-type object whose key value is **cnt_key_value_string**.

Signature

```
(bv-cnt-struct-update <node_path> <cnt_key_value_string>)
```

Parameters

Name	Type	Description
node_path	String	Path to node containing specified field.
cnt_key_value_string	String	Key value of the object whose fields are to be updated.

Returns

Boolean true (**#t**) if the fields update successfully; otherwise, false (**#f**).

Throws

None

Location

bv-cnt-struct-update.monk

Note: *Only those fields whose* **MAP** *subfield has a value of* **1** *are used.*

bv-content-ref-create

Description

Assigns a list of content references to a category.

Signature

```
(bv-content-ref-create <parent_oid> <oid_list>)
```

Parameters

Name	Type	Description
parent_oid	Integer	Object ID of parent category.
oid_list	List	List of object IDs for categories containing desired content.

Returns

Boolean true (**#t**) if the function concludes successfully; otherwise, false (**#f**).

Throws

None

Location

bv-content-ref-create.monk

Example

```
(display (bv-content-ref-create -8043 1 2 3 4 5))(newline)
```

Additional Information

1. Make sure that the products exist before creating content references.
2. If a content reference already exists, it is not created again—but the function still returns **#t**.
3. The list of object IDs can have any number of elements, but *must* have at least one.

bv-content-ref-delete

Description

Removes a list of content references from the **parent_oid**.

Signature

```
(bv-content-ref-delete <parent_oid> <oid_list>)
```

Parameters

Name	Type	Description
parent_oid	Integer	Object ID of parent category.
oid_list	List	List of object IDs for categories containing desired content.

Returns

Boolean true (**#t**) if the function concludes successfully; otherwise, false (**#f**).

Throws

None

Location

bv-content-ref-delete.monk

Example

```
(display (bv-content-ref-create -8043 1 2 3 4 5))(newline)
```

Additional Information

1 Make sure that the products exist before removing content reference.

2 If you attempt to remove a non-existent content reference, nothing is removed, but the function still returns **#t**.

3 The list of object IDs can have any number of elements, but *must* have at least one.

bv-content-ref-list

Description

Retrieves a list of contents belonging to a category, according to the configured parameters.

You can specify "all" or "online" categories.

BroadVision Settings > **Content Status**: **ALL** or **ONLINE**

You can specify the maximum number to retrieve.

BroadVision Settings > **Maximum Content Count**: range: **1 - 86400**, default: **1**.

Signature

```
(bv-content-ref-list <oid>)
```

Parameters

Name	Type	Description
oid	Integer	Object ID of desired category.

Returns

Upon success, a vector of OIDs belonging to the category, for example: **#(8204 8580)**

Throws

None

Location

bv-content-ref-list.monk

Example

```
(display (bv-content-ref-list -8047))(newline)
```

bv-date-to-sap-date

Description

Converts BroadVision date format **mm/dd/yy hh:mm:ss** to SAP date format **yyyymmdd**.

Signature

```
(bv-date-to-sap-date <bv_date_string> <decade>)
```

Parameters

Name	Type	Description
bv_date_string	String	Date in BroadVision format.
decade	String	Determines 20th or 21st Century.

Returns

A string representing the date in SAP format.

Throws

None

Location

bv-util.monk

ewbv-init

Description

Reads in the configuration parameters and establishes the Monk environment.

Signature

```
(ewbv-init)
```

Parameters

None.

Returns

The string **"SUCCESS"** indicates success, **"FAILURE"** indicates failure.

Throws

None

Location

ewbv-init.monk

ewbv-shutdown

Description

Shuts down the BroadVision e*Way.

Signature

```
(ewbv-shutdown)
```

Parameters

None.

Returns

An empty string.

Throws

None.

Location

ewbv-shutdown.monk

sap-date-to-bv-date

Description

Converts SAP date format **yyyymmdd** to BroadVision date format **mm/dd/yyyy**.

Signature

```
(sap-date-to-bv-date <sap_date_string>)
```

Parameters

Name	Type	Description
sap_date_string	String	Date in SAP format.

Returns

A string representing the date in BroadVision format.

Throws

None

Location

bv-util.monk

7.4 Generic e*Way Functions

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

The current set of basic Monk functions is:

event-commit-to-egate on page 102

event-rollback-to-egate on page 103

event-send-to-egate on page 103

event-send-to-egate-ignore-shutdown on page 104

event-send-to-egate-no-commit on page 104

get-logical-name on page 105

insert-exchange-data-event on page 105

send-external-up on page 106

send-external-down on page 106

shutdown-request on page 107

start-schedule on page 107

stop-schedule on page 108

waiting-to-shutdown on page 108

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 104

event-send-to-egate-no-commit on page 104

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 103

event-send-to-egate-no-commit on page 104

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 102

event-rollback-to-egate on page 103

event-send-to-egate on page 103

event-send-to-egate-ignore-shutdown on page 104

get-logical-name

Description

Returns the logical name of the e*Way.

Signature

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

Parameters

None.

Returns

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

Throws

None.

insert-exchange-data-event

Description

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

Signature

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

Parameters

None.

Returns

None.

Throws

None.

See also

Exchange Data Interval on page 66

Zero Wait Between Successful Exchanges on page 67

send-external-up

Description

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

Signature

```
(send-external-up)
```

Parameters

None.

Returns

None.

Throws

None.

send-external-down

Description

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

Signature

```
(send-external-down)
```

Parameters

None.

Returns

None.

Throws

None.

shutdown-request

Description

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

Signature

```
(shutdown-request)
```

Parameters

None.

Returns

None.

Throws

None.

Additional Information

Once interrupted, the e*Way's shutdown cannot proceed until this Monk function is called. If you do interrupt an e*Way shutdown, we recommend that you complete the process in a timely fashion.

start-schedule

Description

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

Signature

```
(start-schedule)
```

Parameters

None.

Returns

None.

Throws

None.

stop-schedule

Description

Requests that the e*Way halt execution of the **Exchange Data with External Function** specified within the e*Way's configuration file. Execution is stopped when the e*Way concludes any open transaction. Does not effect any defined schedules, and does not halt the e*Way process itself.

Signature

```
(stop-schedule)
```

Parameters

None.

Returns

None.

Throws

None.

waiting-to-shutdown

Description

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

Signature

```
(waiting-to-shutdown)
```

Parameters

None.

Returns

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

Throws

None.

Index