SeeBeyond ICAN Suite

C Generic e*Way Extension Kit Developer's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Version 20050405213143.

Contents

Chapter 1

Introduction

The C Generic e*Way Extension Kit enables the developer to extend the client side of e*Gate Integrator by using standard C DLLs. This document describes how to install and configure the C Generic e*Way.

1.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system; to have expert-level knowledge of Windows 2000 and/or UNIX operations and administration; to be thoroughly familiar with and possess an understanding of C and C++ programming languages and Windows-style GUI operations.

1.2 Components

The C Generic e*Way is comprised the following:

- **stcewgenericdll.exe**, the executable component
- **stcewgenericdll.def**, the executable configuration definition file
- **stcewgenericdll.h**, the header file, which is used with the external interface object to define necessary type definitions.
- Method Protoypes, which control the Communications segment of the e*Way.
- Helper functions, which assist with data access.

stcewgenericdll.exe

This executable component, **stcewgenericdll.exe**, is the core of the e*Way that communicates and manipulates Events traveling between an external system and e*Gate, using the C external function scripts. It implements the communication between the external system and e*Gate and loads and interprets the configuration file used by the e*Way to determine how to deal with data to and from the external system.

stcewgenericdll.def

The configuration definition file, **stcewgenericdll.def**, contains all the configuration parameters used by the e*Way executable. Some of these parameters form the basic characteristics for the e*Way itself, while others are functions that allow the e*Way to communicate with a specific external system. The remaining parameters consist of a set of variables used by the environment. These configuration parameters are set using the e*Way Editor.

1.3 Supported Operating Systems

The C Generic e*Way Extension Kit supports the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0 and HP-UX 11i (PA-RISC)
- HP Tru64 V5.1A
- IBM AIX 5.1L and 5.2
- IBM z/OS V1.3 and V1.4
- Red Hat Enterprise Linux AS 2.1 (Intel x86)
- Sun Solaris 8 and 9
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0 and 11i (PA-RISC)
- Japanese Sun Solaris 8 and 9

1.4 System Requirements

To use the C Generic e*Way, you need the following:

- A TCP/IP network connection.
- Additional disk space for e*Way executable, configuration, library, and script files. The disk space is required on both the Participating and the Registry Host. 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.
- Open and review the Readme.txt for the C Generic e*Way regarding any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM at **setup\addons\ewmux**.

1.4.1 Additional Information

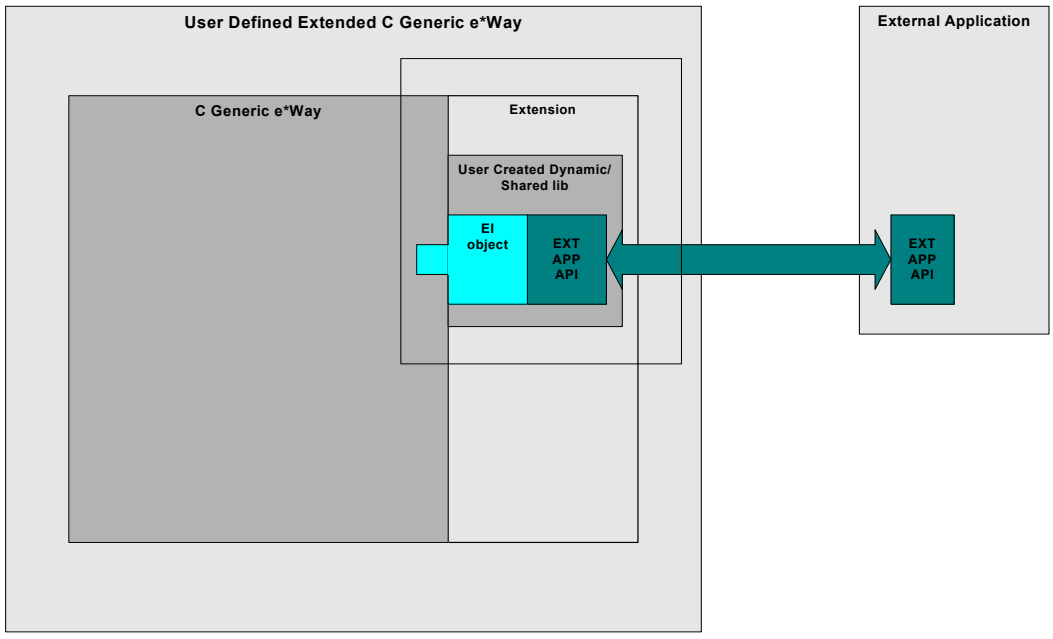
The C Generic e*Way Extension Kit supports the following:

- Solaris SunPro C, version 5.0 for Solaris
- HP "C" Compiler, version 3.15 for HP-UX 11.0
- AIX VisualAge, version 4.0
- Microsoft C, version 6.0
- JBM C89 compiler version 2.10. Makefiles are based on GNUmake.

1.5 e*Way Extensions and External Applications

Figure 1 illustrates how the C Generic e*Way accesses an external application.

Figure 1 Extending the C Generic e*Way



1. A *dynamic link library* or *dll* (on 2000) or *shared* (on UNIX) library is created from user-created source code to extend the C Generic e*Way.
2. The C Generic e*Way is configured to use the user-created dll or shared library.
3. A user-written C script uses the External Interface (EI) protocol and the user-created library to access the external application.

1.5.1 Basics Steps to Extend the C Generic e*Way

To extend the C Generic e*Way for access to an external application, follow these basic steps:

1. Create a dynamic link library or shared library for the C Generic e*Way to use at run-time to access the external application. To do this, create source code in C or C++ using the EI protocol to "wrap" the external application's API calls; then, compile and link the source code to create the dynamic or shared library.
2. Modify the **stcewgenericdll.def** file template as needed to allow proper configuring of the C Generic e*Way with the Configuration GUI. If you do modify the file template, you must import the changed template to the appropriate schema.
3. Write C code that invokes the EI object to enable use of the "wrapped" external application API calls.
4. Configure the C Generic e*Way to use the dynamic link library or shared library you created to extend the C Generic e*Way.
5. Run the extended C Generic e*Way in your e*Gate environment.

Chapter 2

Installation

This chapter describes the following:

2.1 Files/Directories Created by the Installation

The C Generic e*Way Extension Kit includes the files shown in Table 1. The files will be stored within the "egate\client" tree on the Participating Host and committed to the default schema on the Registry Host.

Table 1 : Files created by the e*Gate installation

e*Gate Directory	File(s)
\client\bin	stcewgenericdll.exe
\Server\registry\repository\default\configs\stcewgenericdll	stcewgenericdll.def

2.1.1 sdk.taz Files

The C Generic e*Way Extension Kit requires additional files that are located in **sdk.taz** on the installation CD. It includes the **.lib**, **.dll**, and **.h** header files, which are not installed as part of the Registry Host and Participating Host as part of the e*Gate installation.

You must extract **sdk.taz** to any directory of your choice in your own development environment. The directories and files you must include are listed below.

.h files

You must include the following directories when compiling your DLL:

- `<sdk>/include`
- `<sdk>/include/common/message`
- `<sdk>/include/stc`

In your source files (***.cxx**), the following header files must also be included:

- `<sdk>/include/ewgenericdllfuncs.h`
- `<sdk>/include/tracelog.h`

.lib files (for Windows only)

- <sdk>\lib\win32\stc_common.lib
- <sdk>\lib\win32\stc_stcapis.lib

.dll files

- <sdk>/bin/<operating system name>/stc_common.dll

Chapter 3

Extending the .def File

This chapter describes how to extend the **.def** file and discusses the **.def** file keywords and their arguments. In addition, it also discusses how to test and debug the **.def** file and lists some of the common error messages. It also provides information on configuration parameters and the **.cfg** file.

3.1 Introduction

The C Generic e*Way is configured using the e*Way Editor, a graphical user interface (GUI) that enables you to change configuration parameters quickly and easily. A definition file (**.def**) configures the e*Way Editor to gather those parameters by specifying the name and type of each parameter, as well as other information (such as the range of permissible options for a given parameter). The e*Way Editor stores the values that you assign to those parameters within two configuration files. The configuration files contain similar information but are formatted differently. The **.cfg** file contains the parameter values in delimited records and is parsed by the e*Way at run time. The **.sc** file contains the parameter values and additional information needed by the GUI. The e*Way Editor loads the **.sc** file—not the **.cfg** file— when you edit the configuration settings for an e*Way. Both configuration files are generated automatically by the e*Way Editor whenever the configuration settings are saved.

The **.def** file for the C Generic e*Way contains a set of parameters that are required and may not be modified. You can extend the **.def** file if your modifications to the C Generic e*Way require the definition of user-settable parameters. This chapter discusses the structure of the **.def** and the configuration files and the syntax of the keywords used to configure the e*Way Editor to gather the desired configuration parameters. The e*Way Editor itself is discussed elsewhere; for more information, see the *e*Gate Integrator User's Guide* or the e*Way Editor's Help system.

Important

We strongly recommend that you make no changes whatsoever to the default **stcewgenericdll.def** file. However, you should use that file as a base from which you create your extensions. Save a copy of the file under a unique name and make your changes to the copy.

3.1.1 Layout

The **.def** file has three major divisions:

- The **header** describes basic information about the file itself, such as version number, modification history, and comments.
- The **sub-header** contains several read-only variables that are for internal use only and must not be modified from their default values.
- The **body** contains configuration parameters grouped into sections. Three sections (General Settings, Communications Parameters, and DLL Configuration) must be included in all C Generic e*Way **.def** files; additional sections can be added as needed to support user-created functions.

3.2 .def file Keywords: General Information

All keywords and their arguments are enclosed in balanced parentheses. Keyword arguments can be a quoted string, a quoted character, an integer, a parenthesis-bounded list, a keyword modifier, or additional keywords.

Examples:

```
(name "TCP Port Number" )

(eway-type
    (direction "<ANY">)
)

(set
    (value (1 2 3))
    (config-default (1 2 3))
)

(range
     (value (const 1 const 1024)
)
```

3.2.1 White Space

White space that is not contained within double-quotation marks, including tabs and newlines, is ignored except as a separator between keywords.

For example, the following are equivalent:

- (user-comment (value "") (config-default ""))
- (user-comment
    (value "")
    (config-default "")
)

Whitespace within quotation marks is interpreted literally. For example, **(name "Extra Spaces")** will display as

```
Extra      Spaces
```

in the e*Way Editor's list of names.

3.2.2 Integer Parameters

The maximum value for integer parameters ranges from approximately -2 billion to 2 billion (specifically, -2,147,483,648 to 2,147,483,647). Most ranges will be smaller, such as "1 to 10" or "1 to 1,000."

3.2.3 Floating-point Parameters

Floating-point parameters and floating-point arithmetic are not supported.

3.2.4 String and Character Parameters

String and character parameters may contain all 255 ASCII characters. The "extended" characters are entered using an escaped format:

- Characters such as tab, newline, and carriage return can be entered as **\t**, **\n**, and **\c**, respectively.
- Characters may also be entered in octal or hexadecimal format using **\o** or **\x**, respectively (for example, \x020 for ASCII character 32).

Strings are delimited by double quotes, characters by single quotes. Examples:

- Strings: `"abc" "Administrator"`
- Characters: `'0' '\n'`

Single quotation marks, double-quotation marks, and backslashes that are not used as delimiters (for example, when used within the text of a description) must be escaped with a backslash, as shown respectively below:

- `\'`
- `\"`
- `\\`

3.2.5 Path Parameters

Path parameters can contain the same characters as other string parameters; however, the characters entered should be valid for pathnames within the operating system on which the e*Way runs.

Backslashes in DOS pathnames must be escaped (as in **c:\\home\\egate**).

3.2.6 Comments

Comments within the **.def** file begin with a semi-colon (;). Any semi-colon that appears in column 1, or that is preceded by at least one space character and that does not appear within quotation marks, is interpreted as a comment character.

Examples

```
; this is a valid comment, because it begins in column 1
(name "Section name") ; this is also a valid comment, because it is separated by a space
```

3.2.7 "Header" Information

"Header" information that developers may use to maintain a revision history for the **.def** file is stored within the **(general-info)** section. All the information in this section is maintained by the user; no e*Gate product modifies this information.

Table 2 describes the user-editable parameters in the **(general-info)** section. The use of these fields is not required and they may be left blank, but all the fields must be present. The format and contents of these fields is completely at the developer's discretion, as long as rules for escaped characters are observed (see **"String and Character Parameters" on page 15** for more information). Any **(general-info)** parameters that are not shown in the table below are reserved and should not be modified except by direction of SeeBeyond support personnel.

Table 2 User-editable **(general-info)** parameters

Parameter name	Describes
version	The version number
revision	The revision number
user	The user who last edited the file
modified	The modification date
creation	The creation date
description	A description for this **.def** file, displayed within the e*Way Editor from the File menu's **Tips** option. Quotation marks within the description must be escaped (\").
user-comment	Comments left by the user (rather than the developer), accessed within the e*Way Editor from the File menu's **User notes** option. Unless you wish to provide a default set of "user notes," we recommend you leave this field blank.

3.3 Defining a New Section

The **(section)** keyword defines a section within the **.def** file. The syntax of the new section is described immediately below. Each section requires at least one parameter; see **"Parameter Syntax" on page 18** for more information on defining parameters.

Note: *Section names and parameter names within a section must be unique.*

3.3.1 Section Syntax

Sections within the **.def** file have the following syntax:

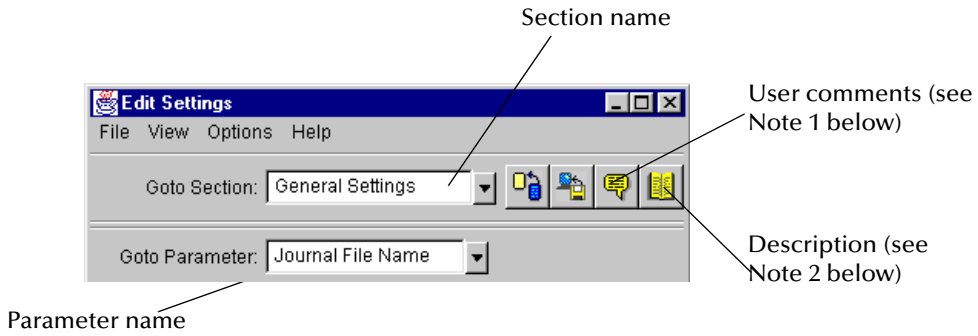
```
(section
  (name "section name")

... at least one parameter definition ...

(description "description text")
(user-comment
    (value "")
    (config-default "")
  ) ; end of user comment
) ; end of section
```

The section name, description text, and user-comment "value" will appear in the e*Way Editor, as shown in Figure 2.

Figure 2 e*Way Editor main controls



Notes

1. The user-comment feature enables users to make notes about a section or parameter that will be stored along with the configuration settings and save those notes along with the configuration settings. Under most circumstances, we recommend that developers leave the **(user-comment)** fields blank, but you can enter information in the **(user-comment)** field if you want to ensure that all user notes for a given section begin with preset information.
2. The description is displayed when the user clicks the "Tips" button. Use this field to create "online help" for a section or parameter. We recommend that you provide a description for every section and every parameter that you create.

3.3.2 Parameter Syntax

Parameters within the **.def** file use the following basic structure:

```
(param-keyword
     (name "Parameter name goes here")
     (value val)
     (config-default val)

     ...additional keywords (range, units, set) as required...

     (description "description text")
     (user-comment
       (value "")
       (config-default "")
     )
  )  ; end of parameter definition
```

The keywords that are invariably required to define a parameter are

- A parameter keyword, discussed below
- The parameter's name: **(name)**
- The initial default value: **(value)**
- The "configuration default": **(config-default)**, which the user can restore by clicking  . This value can be overridden by the **config-default** keyword specified within a **(set)** command; see **"Parameters Accepting a Single Value From a Set" on page 20** and **"Parameters Accepting Multiple Values From a Set" on page 21** for more information.

Note: *The **(value)** keyword is* **always** *followed immediately by the **(config-default)** keyword.*

- The "description" (see the Notes for **"Section Syntax" on page 17** for additional information)
- The "user comment" (see the Notes for **"Section Syntax" on page 17** for additional information), which has its own value and configuration default.

Additional keywords may be required, based upon the parameter keyword and user requirements; these will be discussed in later sections.

Order of Keywords

Keywords must appear in this order:

1. parameter definition*
2. name*
3. value*
4. config-default*
5. set

6 range
7 units
8 show-as
9 factor
10 description*
11 user-comment*

Note: *Keywords marked with * are mandatory for all parameters. The **set** keyword is mandatory for **-set** and **-set-multi** parameters. The remaining keywords (items 6 through 9) are optional and depending on developer requirements may appear in any combination, but they must appear in the above order.*

Parameter Types

There are eight types of parameters. The table below lists the types of parameters that can be defined, the keyword required to define them, and the values that the keyword can accept for the **(value)** and **(config-default)** keywords.

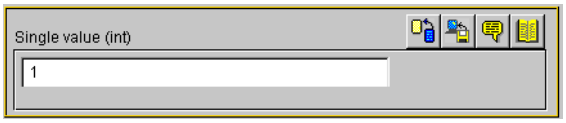
Table 3 Basic parameter keywords

Type	Parameter keyword	Accepts values	Example
Integer	int	integer	7500
Character	char	single-quoted character	'a' '!' '\o123' (octal)
String	string	double-quoted string	“Hello, world”
Date	date	comma-delimited date in *MMM,dd,yyyy* format	AUG,13,2000
Time	time	colon-delimited time in 24-hour *hh:mm:ss* format	15:30:00
Path	path	path; DOS pathnames should use escaped backslashes	/home/egate/client (UNIX) c:\\home\\egate\\client (DOS)
Schedule	schedule	schedule	See **“Schedule Syntax” on page 32**

Parameters Requiring Single Values

Parameters requiring single values are defined within the basic structure shown in **“Parameter Syntax” on page 18**.

Figure 3 A parameter requiring a single value



The parameter is defined using a parameter keyword, as listed in Table 3 on page 19.

Example

To create a parameter that accepts a single integer as input, and to specify "3" as the default and configuration-default value, enter the following:

```
(int
    (name "Parameter requiring a single integer")
    (value 3)
    (config-default 3)
    (description "

      This parameter requires a single integer as input.
")
    (user-comment
      (value "")
      (config-default "")
    )
  )  ; end of parameter definition
```

If you want to limit the values that the user may enter, you may include the optional **(range)** keyword; see "Specifying Ranges" on page 23 for more information.

Parameters Accepting a Single Value From a Set

Adding the suffix **-set** to the basic parameter keyword (**int-set**, **string-set**, **path-set**, and so on) defines a parameter that accepts one of a given list of values.

Figure 4 A parameter requiring one of a set of values

Sets require modifications to the basic parameter syntax (shown in "Parameter Syntax" on page 18):

- An additional required keyword, **(set)**, defines the elements of the set.
- Within the **(set)** keyword, **(value)** and **(config-default)** require arguments within parenthesis-bound lists, as in the following:

```
(value (1 2 3))
(config-default (1 2 3))
```

- To prevent a user from to adding or removing choices from the list you provide, add the **const** keyword to the "value" declaration:

```
(value const (1 2 3))
(config-default (1 2 3))
```

- To specify an empty set, enter the keyword **none**, as below:

```
(value none)
(config-default none)
```

Note: *"-set-multi" keywords use a different syntax to define an empty set; see* **"Parameters Accepting Multiple Values From a Set" on page 21** *for more information.*

Other important considerations:

- The value specified as the initial **(value)** for the parameter must match at least one of the the values specified for **(config-default)** within the **(set)** keyword.
- The initial value within the **(set)** keyword's **(config-default)** list must be within the **(set)** keyword's **(value)** list. However, we strongly recommend that you simply make the two lists identical.

Example

To create a parameter that accepts one of a fixed set of integers (like the one shown in Figure 4 above), enter the following:

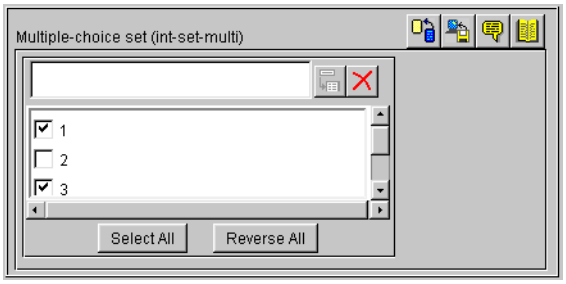
```
(int-set
    (name "Single-choice set (int-set)")
    (value 1)
    (config-default 1)
    (set
      (value const (1 2 3))
      (config-default (1 2 3))
    )
    (description "Provides a single choice from a list of integers.")
    (user-comment
      (value "")
      (config-default "")
    )
 )  ; end of int-set
```

Note: *The values specified by the* ***(set)*** *keyword must be within any values specified by the* ***(range)*** *keyword. See* **"Specifying Ranges" on page 23** *for more information.*

Parameters Accepting Multiple Values From a Set

Adding the suffix **-set-multi** to the basic parameter keyword (**int-set-multi**, **string-set-multi**, **path-set-multi**, and so on) defines a parameter that accepts one or more options from a given list of values.

Figure 5 A parameter requiring one of a set of values



Sets require modifications to the basic parameter syntax (shown in **"Parameter Syntax" on page 18**):

- An additional required keyword, **(set)**, defines the elements of the set.
- Within the **(set)** keyword, **(value)** and **(config-default)** require arguments within parenthesis-bound lists, as in the following:

```
(value (1 2 3))
(config-default (1 2 3))
```

- To prevent a user from to adding or removing choices from the list you provide, add the **const** keyword to the "value" declaration:

```
(value const (1 2 3))
(config-default (1 2 3))
```

- To specify an empty set, enter an empty pair of parentheses "()", as below:

```
(value () )
(config-default () )
```

  "-set" keywords use a different syntax to define an empty set; see **"Parameters Accepting a Single Value From a Set" on page 20** for more information.

Other important considerations:

- The value specified as the initial **(value)** for the parameter must match at least one of the the values specified for **(config-default)** within the **(set)** keyword.
- The initial value within the **(set)** keyword's **(config-default)** list must be within the **(set)** keyword's **(value)** list. However, we strongly recommend that you simply make the two lists identical.

Example

To create a parameter that accepts one of a fixed set of integers (like the one shown in Figure 5 above), enter the following:

```
(int-set-multi
    (name "Multiple-choice set (int-set-multi)")
    (value (1 3))
    (config-default (1 3))
    (set
      (value (1 2 3 4 5))
      (config-default (1 2 3 4 5))
    )
    (description "Integer with a modifiable multiple-option set")
    (user-comment
      (value "")
      (config-default "")
    )
  )  ; end of int-set-multi
```

Note: *The order in which keywords appear is very important. See* **"Order of Keywords" on page 18** *for more information.*

3.3.3 Specifying Ranges

The **(range)** keyword enables you to limit the range of options that the user may input as a parameter value for **int** and **char** parameters. You may specify a fixed range, or allow the user to modify the upper limit, the lower limit, or both limits. Range limits are inclusive. The values you specify as limits indicate the lowest or highest acceptable value.

The syntax of (range) is as follows:

```
(range
      (value ([const] lower-limit [const] upper-limit))
      (config-default (lower-limit upper-limit))
    )
```

The optional **const** keyword specifies that the limit is fixed; if the keyword is omitted, the limit can be modified by the user. The **const** keyword must precede each limit if both limits are to be fixed.

Example

This example illustrates how to define a parameter that accepts an integer as input and limits the range of legal values from zero to ten.

```
(int
    (name "Single integer with fixed range")
    (value 5)
    (config-default 5)
    (range
      (value (const 0 const 10))
      (config-default (0 10))
    )
    (description "Accepts a single integer, limited to a fixed
range.")
    (user-comment
      (value "")
      (config-default "")
    )
  ) ; end of int parameter
```

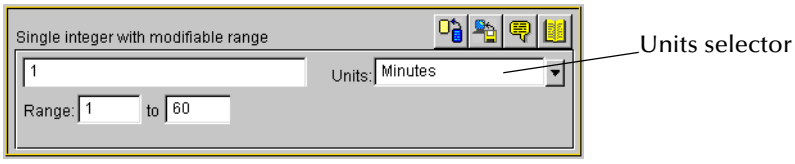
You may also use **(range)** to specify a character range; for example, a range of "A to Z" would limit input to uppercase letters, and a range of "! to ~" limits input to the standard printable ASCII character set (excluding space).

Note: *You may also specify ranges for* ***-set*** *and* ***-set-multi*** *parameters (****int-set****,* ***char-set****, and so on).*

3.3.4 Specifying Units

The **(units)** keyword enables **int** parameters to accept input and display the list of available options in different units, provided that each unit is an integer multiple of a base unit.

Figure 6 A parameter that performs unit conversion



Acceptable groups of units include

- Seconds, minutes, hours, days
- Bytes, kilobytes, megabytes

Unit conversions that require floating-point arithmetic are not supported.

The syntax of the **(units)** keyword is

```
(units
     ("base-unit":1 "first-unit":a "second-unit":b ... "nth-unit":n)
          (value "default-unit")
          (config-default "default-unit")
     )
```

where ***a***, ***b***, and ***n*** are the numbers by which the base unit size should be multiplied to perform the conversion to the respective units. The base unit should normally have a value of 1, as shown above; while the e*Way Editor will permit other values, it is highly unlikely that an application would require any other number. The units themselves have no meaning to the e*Way Editor other than the relationships you define (in other words, the Editor does not identify or process "seconds" or other common units as such).

Example

To specify a set of time units (seconds, minutes, hours, and days), enter the following:

```
(units
     ("Seconds":1 "Minutes":60 "Hours":3600 "Days":86400)
          (value "Seconds")
          (config-default "Seconds")
    )
```

Units, Default Values, and Ranges

Any time you use the **(units)** keyword within a parameter, you must make sure that the default values can be expressed as integer values of *each* unit. Observing this principle prevents end users from receiving error messages when changing e*Way Editor values in a specific order. For example, if you specified the time units in the example above, but assigned the parameter a default value of "65 seconds," any user who selects the minutes unit *without changing the default value* will receive an error message, because the e*Way Editor cannot convert 65 seconds to an integral number of minutes. Ranges, however, will be rounded to the nearest integer.

Note: *No matter what default value you specify, a user will always see an error message if an inconvertible value is entered and the unit selector is changed. We recommend that you design your parameters so that error messages are not displayed when* ***default*** *values are entered.*

Example

To define a time parameter that displays values in seconds or minutes, with a default of 120 seconds and a fixed range of 60 to 3600 seconds (1 minute to 60 minutes), enter the following:

```
(int
    (name "Single integer with fixed range")
    (value 120)
    (config-default 120)
    (range
      (value (const 60 const 3600))
      (config-default (60 3600))
    )
    (units
      ("Seconds":1 "Minutes":60)
      (value "Seconds")
      (config-default "Seconds")
    )
    (description "Accepts a value between 1 and 60 minutes, with
                 a default units value in seconds.")
    (user-comment
      (value "")
      (config-default "")
    )
  ) ; end parameter
```

Note: *The order in which keywords appear is very important. See* **"Order of Keywords" on page 18** *for more information.*

3.3.5 Displaying Options in ASCII, Octal, Hex, or Decimal

The **(show-as)** keyword enables you to create **int** or **char** parameters that a user can display in ASCII, octal, hexadecimal, or decimal formats.

The syntax of the **(show-as)** keyword is

```
(show-as
      (format-keyword1 [format-keyword2 ... format-keywordn])
      (value format-keyword)
      (config-default format-keyword)
)
```

where ***format-keyword*** is one of the following:

- ascii
- octal
- hex
- decimal

Format keywords are case-insensitive, and may be used in any combination and in any order.

Be sure that any default values you specify for a parameter that uses **(show-as)** can be represented in each of the **(show-as)** formats. For example, if you are using **(show-as)** to show an integer parameter in both decimal and hex formats, the default value must be non-negative.

Example

To create a parameter that accepts a single character in the character-code range between 32 and 127 and that can display the character value in ASCII, hex, or octal, enter the following:

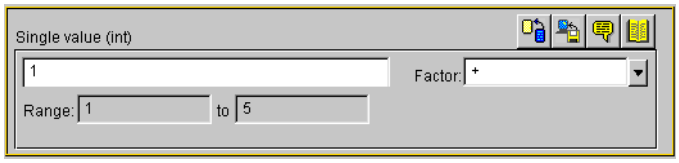
```
(char
    (name "A single ASCII character")
    (value '\o100')
    (config-default '\o100')
    (range
      (value (const '\o040' const '\o177'))
      (config-default ('\o040' '\o177'))
    )
    (show-as
      (Ascii Octal Hex)
      (value Octal)
      (config-default Octal)
    )
    (description "Accepts a single character between ASCII 32 and ASCII 127.")
    (user-comment
      (value "")
      (config-default "")
    )
 ) ; end char parameter
```

Note: *The order in which keywords appear is very important. See* **"Order of Keywords" on page 18** *for more information.*

Factor

The **(factor)** keyword enables users to enter an arithmetic operator (+, –, *, or /) as part of an **int** parameter; for example, to indicate that a value should increase by five units, the user would enter the integer "5" and the factor "+".

Figure 7 A parameter using **(factor)**



The syntax of the **(factor)** keyword is

```
(factor
     ('operator1' ['operator2'... 'operatorN'])
     (value 'operator'
     (config-default 'operator')
)
```

where ***operator*** is one of the four arithmetic operators +, –, *, or / (forward slash).

Example

To define a parameter that accepts an integer between 1 and 5 with a factor of + or – (as in Figure 7 above), enter the following:

```
(int
     (name "Integer with factor")
     (value 1)
     (config-default 1)
     (range
          (value (const 1 const 5))
          (config-default (1 5))
```

```
        )
        (factor
            ('+' '-')
            (value '+')
            (config-default '+')
        )
        (description "Enter an integer between 1 and 5 and a factor of +
or -.")
        (user-comment
            (value "")
            (config-default "")
        )
)    ; end int parameter
```

Note: *The **(factor)** keyword must be the final keyword before the **(description)** keyword. See* **"Order of Keywords" on page 18** *for more information.*

Encrypting Strings

Encrypted strings (such as for passwords) are stored in string parameters; to specify encryption, use the **encrypt** keyword, as in the following:

```
(string encrypt
    ...additional keywords follow...
```

The e*Way Editor uses the parameter that immediately precedes the encrypted parameter as its encryption key; therefore, be sure that the parameter that prompts for the encrypted data is not the first parameter in a section. The easiest way to accomplish this is to define a "username" parameter that immediately precedes the encrypted "password" parameter. If you need to specify an encryption key other than the user name, you must define a separate parameter for this purpose.

Text entered into an encrypted-string parameter is displayed as asterisks ("***").

Example

To create a password parameter, enter the following *immediately following* the parameter definition for the corresponding user name (not shown):

```
(string encrypt
    (name "Password")
    (value "")
    (config-default "")
    (description "The e*Way Editor will encrypt this value.")
    (user-comment
      (value "")
      (config-default "")
    )
 )
```

Note: *The **encrypt** keyword can only follow the **string** keyword. The only parameter type that can be encrypted is **string**; integer, character, path, time, date, or schedule parameters cannot be encrypted.*

3.4 Configuration Keyword Reference

Table 4 lists the keywords that may appear in the **.def** file.

Table 4 .def-file keywords

Keyword	Purpose	For more information, see this section
app-protocol	Reserved; do not change from the default "<ANY>".	
cfg-icon	Reserved; do not change from the default "" (null string).	
char	Declares a character parameter	"Parameter Types" on page 19
char-set	Declares a set of characters, one of which must be selected (via radio button)	"Parameters Accepting a Single Value From a Set" on page 20
char-set-multi	Declares a set of characters, any of which may be selected (via check boxes)	"Parameters Accepting Multiple Values From a Set" on page 21
config-default	Specifies the values that will be restored when the user clicks the e*Way Editor's  button	"Parameter Syntax" on page 18
const	Specifies that a value cannot be changed by the user	"Specifying Ranges" on page 23
creation	Records creation date or other information.	""Header" Information" on page 16
date	Declares a date parameter	"Parameter Types" on page 19
date-set	Declares a set of dates, one of which must be selected (via radio button)	"Parameters Accepting a Single Value From a Set" on page 20
date-set-multi	Declares a set of dates, one of which must be selected (via radio button)	"Parameters Accepting Multiple Values From a Set" on page 21
delim1	Defines the line-separator delimiter used within .cfg files. We recommend that you do not modify this value.	
delim2	Defines the parameter-name delimiter used within .cfg files. We recommend that you do not modify this value.	
delim3	Defines the value-separating delimiter used within .cfg files. We recommend that you do not modify this value.	

Table 4 .def-file keywords

Keyword	Purpose	For more information, see this section
delim4	Defines the list-item-separating delimiter used within .cfg files. We recommend that you do not modify this value.	
description	A description for the entry (displayed using the e*Way Editor's  button	"Notes" on page 17
direction	Reserved; do not change from the default "<ANY>".	
encrypt	Encrypts a string, such as for passwords. Valid only after the **string** keyword.	"Encrypting Strings" on page 28
factor	Defines an arithmetic operator to be associated with an integer parameter	"Factor" on page 27
general-info	Defines the "general information" division of the **.def** file	""Header" Information" on page 16
generated-cfg-path	Specifies the path in which the .cfg file will be stored. We recommend that you do not modify this field.	
int	Declares an integer parameter	"Parameter Types" on page 19
int-set	Declares a set of integers, one of which must be selected (via radio button)	"Parameters Accepting a Single Value From a Set" on page 20
int-set-multi	Declares a set of integers, any of which may be selected (via check boxes)	"Parameters Accepting Multiple Values From a Set" on page 21
modified	Records modification date or other information	""Header" Information" on page 16
name	Specifies the name of a parameter or a section	"Parameter Syntax" on page 18
network-protocol	Reserved; do not change from the default "<ANY>".	
os-platform	Reserved; do not change from the default "<ANY>".	
path	Declares a path parameter	"Parameter Types" on page 19
path-set	Declares a set of paths, one of which must be selected (via radio button)	"Parameters Accepting a Single Value From a Set" on page 20
path-set-multi	Declares a set of paths, any of which may be selected (via check boxes)	"Parameters Accepting Multiple Values From a Set" on page 21

Table 4 .def-file keywords

Keyword	Purpose	For more information, see this section
protocol-api-version	Reserved; do not change from the default "<ANY>".	
range	Specifies a range of values that represent the upper and lower limits of acceptable user input	**"Specifying Ranges" on page 23**
revision	Records revision numbering or other information (entered manually by the developer)	**""Header" Information" on page 16**
schedule	Declares a schedule parameter	**"Parameter Types" on page 19** and **"Schedule Syntax" on page 32**
schedule-set	Declares a set of schedules, one of which must be selected (via radio button)	**"Parameters Accepting a Single Value From a Set" on page 20** and **"Schedule Syntax" on page 32**
schedule-set-multi	Declares a set of schedules, any of which may be selected (via check boxes)	**"Parameters Accepting Multiple Values From a Set" on page 21** and **"Schedule Syntax" on page 32**
section	Defines a "section" of the **.def** file	See **"Section Syntax" on page 17**
set	Defines the elements in a set	**"Parameters Accepting a Single Value From a Set" on page 20** and **"Parameters Accepting Multiple Values From a Set" on page 21**
show-as	Selects the format in which character or integer parameters will be displayed	**"Displaying Options in ASCII, Octal, Hex, or Decimal" on page 26**
string	Declares a string parameter	**"Parameter Types" on page 19**
string-set	Declares a set of strings, one of which must be selected (via radio button)	**"Parameters Accepting a Single Value From a Set" on page 20**
string-set-multi	Declares a set of strings, any of which may be selected (via check boxes)	**"Parameters Accepting Multiple Values From a Set" on page 21**
super-client-type	Reserved; do not change from the default "<ANY>".	
time	Declares a time parameter	**"Parameter Types" on page 19**

Table 4 .def-file keywords

Keyword	Purpose	For more information, see this section
time-set	Declares a set of times, one of which must be selected (via radio button)	"Parameters Accepting a Single Value From a Set" on page 20
time-set-multi	Declares a set of times, any of which may be selected (via check boxes)	"Parameters Accepting Multiple Values From a Set" on page 21
units	Determines in which units a parameter will be displayed	"Specifying Units" on page 24
user	Records the name of the user who last edited the file (entered manually by the developer)	""Header" Information" on page 16
user-comment	Records a general comment to be applied to the file (accessible via the e*Way editor)	"Notes" on page 17
value	Defines the initial value for a parameter	"Parameter Syntax" on page 18
version	Records the name of the user who last edited the file (entered manually by the developer)	""Header" Information" on page 16

3.4.1 Schedule Syntax

Schedules can be time-based (as in "every ten minutes" or "every hour"), or calendar-based (for a daily, weekly, monthly, or yearly schedule). The syntax for specifying schedules as values and configuration defaults appears in the table below (all times are specified in 24-hour format):

Table 5 Schedule syntax

For this schedule…	…use this syntax	Example
Every ***s*** seconds	***s*** (s=seconds)	1800 (every 1800 seconds, or every 30 minutes)
Number of seconds after the minute	:::::***s*** (s=seconds)	:::::10 (every ten seconds after the minute)
Number of minutes and seconds past the hour	::::***m***:***s*** (m=minutes; s=seconds)	::::15:00 (every fifteen minutes and zero seconds after the hour)
Daily at *time*	:::***hh***:***mm***:***ss***	:::12:00:00 (daily at noon)

Table 5 Schedule syntax

For this schedule…	…use this syntax	Example
Weekly at *day-of-week* at *time*	::***DD*:*hh*:*mm*:*ss*** (DD=day of week)	::Su:12:00:00 (weekly, Sundays at noon)
Monthly, every *nth day-of-week* at *time*	::***DDn*:*hh*:*mm*:*ss*** (DD=day of week; n=1, 2, 3, 4, or 5)	::Su1:12:00:00 (monthly, the first Sunday, at noon)
Monthly, every *nth* day at *time*	::***n*:*hh*:*mm*:*ss*** (n=day of month)	::3:12:00:00 (monthly, the third day of the month, at noon)
Yearly, at a given *date* at *time*	:***MM*:*dd*:*hh*:*mm*:*ss*** (MM=month; dd=day)	:08:13:04:00:00 (every August 13th at 4:00 AM)
Yearly, every *nth* day of *month* at *time*	:***MM*:*DDn*:*hh*:*mm*:*ss*** (MM=month; DD=day of week; n=1, 2, 3, 4, or 5)	:05:We3:12:00:00 (yearly, every third Wednesday of May, at noon)

Defining Default Schedules

It is *significantly* simpler to define schedules using the e*Way Editor than it is to create schedule entries manually within the **.def** file, especially for complex schedules. The only reason to define a schedule within a **.def** file is to establish a default schedule. If you want to create a default schedule entry, and do not want to construct the entry manually, use this procedure:

1. Define a schedule parameter with a blank ("") default.
2. Commit the **.def** file to a schema, and use the e*Gate Editor to define an entry for the **Start Exchange Data Schedule** parameter. In this entry, create the schedule that you eventually wish to use as a default. (Don't be concerned if this is not the parameter for which you want to define a default schedule; this is just a temporary file.)
3. Save the configuration as **temp** (do not specify an extension) and exit the e*Way Editor.
4. Pull down the Schema Designer's File menu and select **Edit File**.
5. Use the file-selection controls to open the file **/configs/stcewgenericcdll/temp.cfg**.
6. The Notepad editor will launch. Scroll down until you find the "Communications Setup" section; a sample appears below.

```
# ------------------------------------------------------------
#   Section:Communication Setup
# ------------------------------------------------------------
#
Communication Setup|Exchange Data
Interval|value=120|set=120|range=0,86400
```

Schedule definition

7. Use "copy and paste" to copy the schedule-definition string (in the figure above, "`:::12:00:00`").

8 Exit the editor; there is no need to save the file.

9 Pull down the Schema Designer's File menu and select **Edit File**.

10 Use the file-selection controls to open the file **/configs/stcewgenericcdll/*your_def_file*** (substituting the name of the **.def** file you want to modify).

11 Modify the (value) and (config-default) keywords within the desired schedule parameter by pasting in the string that you copied in step 7 above.

12 Save the file and commit the modified file to the Registry (see **"Editing a .def File Within a Schema" on page 97** for more information).

3.5 Configuration Parameters and the Configuration Files

Parameters defined within the **.def** file are stored within two "configuration" files (**.cfg** and **.sc**), which are generated by the e*Way Editor's "Save" command. The following rules apply to both **.cfg** and **.sc** files:

- Keywords are not case sensitive, as they are converted to uppercase internally before matching.
- Comments begin with the "#" character, which must appear in column one (see the example in the section immediately below).
- Unlike the **.def** file, the **.cfg** and **.sc** files are sensitive to white space. White space consists of single space characters, tabs, and newlines. Be careful not to insert extra white space around delimiters or equal signs (for example "|value=3|" is legal, but "|value = 3|" and "| value=3 |" are illegal).

The following rule applies only to the **.cfg** file:

- Each line and each element in the **.cfg** file is separated using delimiters (see **delim1, delim2, delim3, and delim4** in **Table 4 on page 29**). We strongly recommend that you do not modify any of the default delimiters.

Note: *The e*Way Editor will create a **.cfg** and **.sc** file automatically when you save your configuration changes in the e*Way Editor. You should not need to modify either file manually unless directed to do so by SeeBeyond support personnel.*

*Although e*Ways are shipped with default **.def** files, no configuration files are provided, because there is no "standard" configuration for any given e*Way. Users must manually create a configuration profile using the e*Way Editor for every e*Way component.*

Examples

.cfg File

This example is excerpted from the "General Settings" section of a **.cfg** file that is generated by the default **stcewgenericcdll.def** file.

```
# ------------------------------------------------------------------
```

```
#       Section: General Settings
# ------------------------------------------------------------------------
#
General Settings|Journal File Name|value=|set=
General Settings|Max Resends Per Message|value=5|set=5|range=1,1024
General Settings|Max Failed Messages|value=3|set=3|range=1,1024
General Settings|Forward External Errors|value=NO|set=NO,YES
```

.sc File

This example is excerpted from the "General Settings" section of a **.sc** file that is generated by the default **stcewgenericcdll.def** file. Notice the amount of additional information as compared to the **.cfg** file example of the same section above.

```
; ------------------------------------------------------------------------
;   Section: "General Settings"
; ------------------------------------------------------------------------
(section
  (name "General Settings")
  (string-set
    (name "Journal File Name")
    (value "")
    (config-default "")
    (set
      (value (""))
      (config-default (""))
    )
    (description "
 Journal File is used for the following conditions:
 - Journal a message when it exceeds the number of retries.
 - Journal an external error when it's not configured to
   forward to Egate.

 If an absolute path is not specified,  the system data
 directory is prepended to the path.
")
    (user-comment
      (value "")
      (config-default "")
    )
  )
  (int-set
    (name "Max Resends Per Message")
    (value 5)
    (config-default 5)
    (set
      (value (5))
      (config-default (5))
    )
    (range
      (value (const 1 const 1024))
      (config-default (1 1024))
    )
    (description "Max Resends Per Message:

 This parameter is the maximum number of times the e*Way
 will attempt to resend a message to the extenal after
 receiving an error.  When this maximum is reached, the
 message is considered a failed message and is written to
 a journal file.
")
    (user-comment
      (value "")
      (config-default "")
```

```
      )
    )
    (int-set
      (name "Max Failed Messages")
      (value 3)
      (config-default 3)
      (set
        (value (3))
        (config-default (3))
      )
      (range
        (value (const 1 const 1024))
        (config-default (1 1024))
      )
      (description "Max Failed Messages:

 This parameter is the maximum number of failed messages
 the e*Way will allow.  If this many messages fail
 and are journaled, the e*Way will shutdown and exit.
")
      (user-comment
        (value "")
        (config-default "")
      )
    )
    (string-set
      (name "Forward External Errors")
      (value "NO")
      (config-default "NO")
      (set
        (value const ("NO" "YES"))
        (config-default ("NO" "YES"))
      )
      (description "Forward External Errors:

 If this parameter is set to YES then error messages that
 starts with DATAERR received from the external will be
 queued to the configured queue.  If this parameter is set
 to NO then error messages will not be forward.
")
      (user-comment
        (value "")
        (config-default "")
      )
    )
    (description "General Settings:

 This section contains a set of top level parameters:

        o Journal File Name
        o Max Resends Per Message
        o Max Failed Messages
        o Forward External Errors
")
    (user-comment
      (value "")
      (config-default "")
    )
  )
```

3.6 Testing and Debugging the .def File

Testing the **.def** file is very straightforward; simply open the file with the e*Way Editor. If the syntax of all parameters is correct, the e*Way Editor will launch, and you can confirm that your sections, parameters, ranges, and options are as you intended.

There are two types of errors that you may encounter:

- Logical errors: The e*Way Editor will load the **.def** file and will display no error message, but the parameters are not defined as desired (for example, default options are omitted, or a range was not properly defined). These errors are corrected simply by replacing the undesired values with the desired ones.
- Syntax errors: These "mechanical" errors involve missing parentheses, invalid keywords and similar problems. These errors will cause the e*Way Editor to display an error message and exit. This section deals primarily with errors of this type.

Note: *You may also encounter syntax errors if you try to edit an existing configuration profile that contains a corrupted **.sc** file. You should not attempt to modify **.sc** or **.cfg** files outside of the e*Way Editor unless specifically instructed to do so by SeeBeyond personnel.*

The e*Way Editor component that interprets the **.def** file provides only elementary error messages when it encounters an error in the **.def** file. This section discusses the most common errors you may encounter, and the steps you should take to debug a **.def** file under development.

By far, the most common errors are

- Missing parentheses. Proper indentation will help you catch most of these, and some editors have features that find matching parentheses (such as the **vi** editor's SHIFT+% function).
- Missing quotation marks. Be sure that characters are delimited by single quotes and strings/paths by double quotes.
- Quotation marks that should be escaped but are not. This usually occurs in the argument to the **(description)** keyword; double-check that all quotations within descriptions use \"escaped\" quotation marks.
- Missing parameters. Refer to the examples in this chapter, or to the sample **.def** file for the required parameters for each keyword.
- Keywords out of order. See **"Order of Keywords" on page 18**.

Note: *Using the templates provided in the sample **.def** file will help prevent many errors before they occur; see **"Sample .def File" on page 39** for more information.*

3.6.1 Common Error Messages

The following section contains common error messages and their most common causes. Each error message will contain the string **L<*nnn*>**, which indicates a line number (for example, **L<124>** signifies "line 124").

SCparse : parse error, expecting `LP_*keyword-name*'
The keyword ***keyword-name*** was expected but not found. The keyword could be missing or out of order, the keyword's initial parenthesis could be missing, or the previous keyword could have been terminated prematurely (for example, by an out-of-place parenthesis or quote-parenthesis combination) or misspelled.

SCparse : parse error, expecting `RIGHT_PAREN'
The right parenthesis is missing, a close-quote is missing, as in **(user-comment ")**, or there is an extra or unescaped close-quote within a **(description)** keyword argument.

SCparse : parse error, expecting `LEFT_PAREN'
This error appears under a very wide range of conditions. A keyword could be misspelled, there could be extraneous or unbalanced quotes or parentheses, a keyword could be missing a left parenthesis, or extraneous material may have been found between parameter declarations. Sometimes this error appears in conjunction with **expecting `LP_*keyword-name*'**.

***Param-Type*<*keyword*>: Value is not within the allowed range.**
An argument to a keyword has exceeded the limits defined by its accompanying **(range)** keyword. Change either the **(value)** argument or the **(range)** limit.

***param-type*TypeSet<*keyword*> : "*n*" is not in this Set.**
A default value for a parameter has been specified that does not appear within the default value of the **(set)** keyword.

SCparse : parse error, expecting `*arg-type*'
One type of argument was expected, but another has been found (for example, an integer where as string was expected). Errors expecting LITERAL_STRING are commonly caused by missing quotation marks. Errors expecting TIME_VAL, DATE_VAL, or SCHEDULE_VAL can also be due to invalid data (such as a time of 12:00:99), or missing/extra delimiters.

CharVal : "*sequence*" is not legal character.
There is an error in an escape sequence.

SCparse : parse error
This "general" error can be caused by a number of problems, such as misspelled arguments within keywords.

3.7 Sample .def File

A **.def** file containing commented samples of a wide range of parameter definitions is available on the e*Gate installation CD-ROM:

/samples/sdk/ewgendllextfile.def

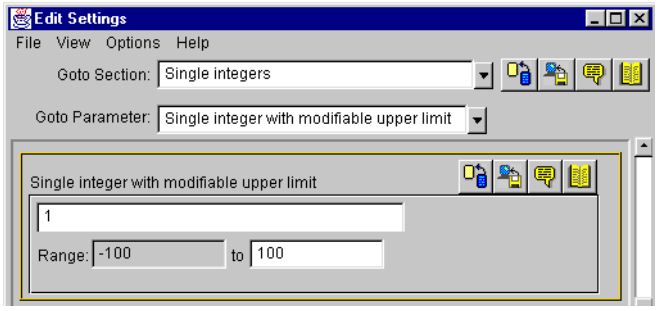
You can use the **ewgendllextfle.def** file as a template from which you can build your own extensions to your own **.def** file. Simply open the file with a text editor, select the desired parameter-definition template, and "copy and paste" the template into your own **.def** file, where you can modify it as needed.

To open the sample.def file in the e*Way Editor:

1. Using the Schema Designer, commit the **ewgendllextfile.def** file to the directory **/configs/stcewgenericdll/** within any desired schema. We recommend that you do not commit the file to the **default** schema; rather, use a schema reserved for testing and development.
2. Create or select an e*Way, and display its properties. Remember that this e*Way cannot be used to manipulate data; it serves merely as a "placeholder" so you can open the **ewgendllextfile.def** file with the e*Way Editor.
3. On the e*Way property sheet's General tab, under **Executable file**, click **Find**.
4. Select **stcewgenericdll.exe** and click **OK**.
5. Under **Configuration file**, click **New**.
6. From the list of e*Way templates, select **ewgendllextfile**.

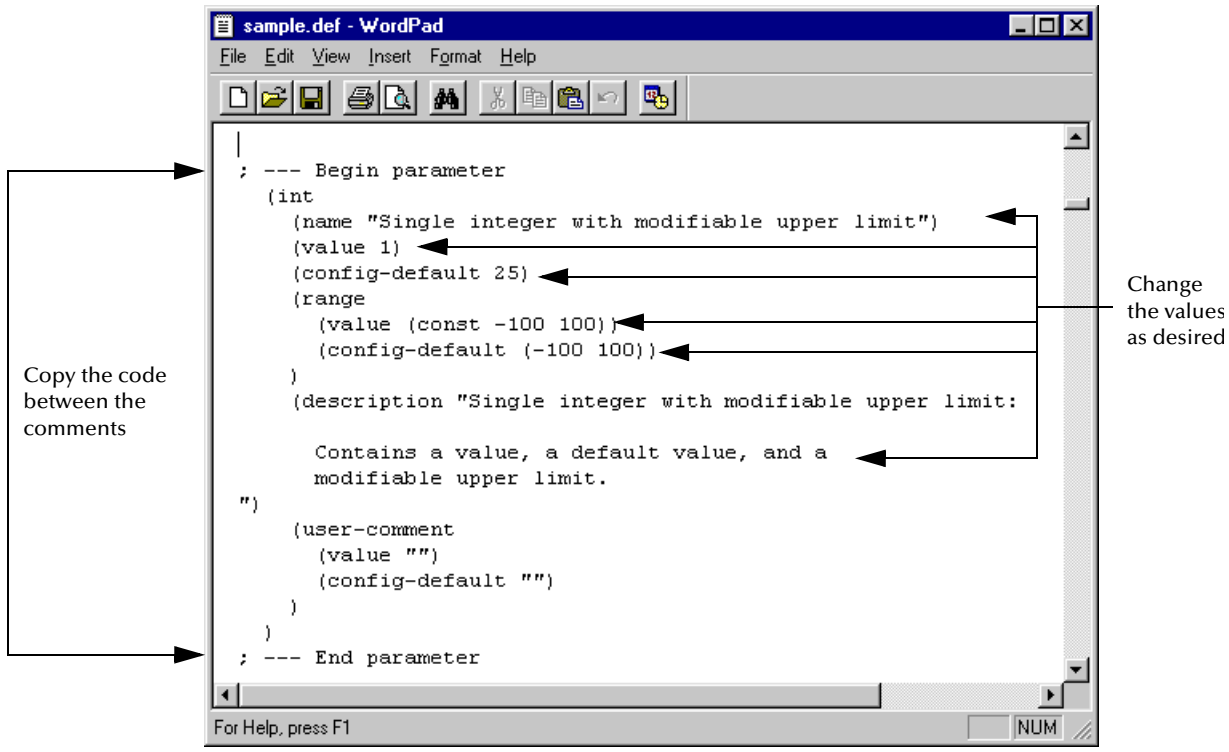
When the e*Way Editor launches, you will see several sections of sample parameters (for example, "Single integer with modifiable lower limit," "Single integer with modifiable upper limit," and so on), as shown in Figure 8.

Figure 8 The **ewgendllextfile.def** file in the e*Way Editor



After identifying the parameter you wish to copy, open **sample.def** in a text editor and search for the parameter name. Then, simply copy the parameter and change the sample values to the values you wish to use, as shown in Figure 9.

Figure 9 The **ewgendllextfile.def** file in Wordpad



3.8 Accessing Configuration Parameters Within the APIs

TheC Generic e*Way automatically loads configuration parameters stored in the **.cfg** file into variables within the APIs.

3.8.1 Variable-name Format

Variables are named using the format

SECTION-NAME_PARAM-NAME

where ***SECTION-NAME*** is the name of the section and ***PARAM-NAME*** is the name of the parameter. The value of the parameter is stored as the value of the variable.

Variable names are in all upper case, and are case-sensitive. The section and parameter names are separated by an underscore, and any spaces contained within section or parameter names are also converted into underscores.

Examples

The value of the parameter named "Password" within the section "Authentication" would be stored in the variable "AUTHENTICATION_PASSWORD" (all upper case).

The value of the parameter named “Gateway ID” within the section “Connection Parameters” would be stored in the variable “CONNECTION_PARAMETERS_GATEWAY_ID”.

3.8.2 Getting Variable Values

Variable values are read using the helper function **GenericEWayHelperGetConfigVariable**. The **GenericEWayHelperGetConfigVariable** retrieves the configuration parameters that the e*Way extracted form the corresponding configuration file.

Examples

```
DWORD cb;
char szFile[MAX_PATH];

cb = MAX_PATH;
if (!(GenericEWayHelperGetConfigVariable(hExt,
                 "File Settings",
                 "To eGate File Name",
                 &cb,
                 szFile,
                 0,
                 NULL)))
{
    TRACE_0(TRACE_CONFIGURATION, TRACE_EVENT_FATAL,
"unable to get setting: File Settings - To eGate File Name");

    goto ParamError;
}
...
```

See **GenericEWayHelperGetConfigVariable** on page 75 for more information.

Chapter 4

Configuration

This chapter describes how to configure the C Generic e*Way.

4.1 e*Way Configuration Parameters

e*Way configuration parameters are set using the e*Way Editor.

To change e*Way configuration parameters:

1. In the Schema Designer's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
3. 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.

For more information about how to use the e*Way Editor, see the e*Way Editor's online Help or the *e*Gate Integrator User's Guide*.

The e*Way's configuration parameters are organized into the following sections:

- General settings
- Communication Setup
- DLL Configuration

4.1.1 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 will be 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 will be journaled for the following conditions:

- When the number of resends is exceeded (see Max Resends Per Message below)
- When its receipt is due to an external error, but Forward External Errors is set to **No**. (See **"Forward External Errors" on page 43** for more information.)

Max Resends Per Message

Description

Specifies the number of times the e*Way will attempt to resend a message (Event) to the external system after receiving an error. When this maximum is reached, the message is considered "Failed" and is written to the journal file.

Required Values

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

Max Failed Messages

Description

Specifies the maximum number of failed messages (Events) that the e*Way will allow. When the specified number of failed messages is reached, the e*Way will shut down and exit.

Required Values

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

Forward External Errors

Description

Selects whether error messages that begin with the string "GEW_FAILURE_DATAERR" that are received from the external system will be queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 56** for more information.

Required Values

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

See **"Schedule-driven Data Exchange Functions" on page 51** for information about how the e*Way uses this function.

4.1.2 Communication Setup

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

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

Exchange Data Interval

Description

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

Required Values

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

Additional Information

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

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

See **"Down Timeout" on page 45** and **"Stop Exchange Data Schedule" on page 45** for more information about the data-exchange schedule.

Zero Wait Between Successful Exchanges

Description

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

Required Values

Yes or **No**. If this parameter is set to **Yes**, the e*Way will immediately invoke the **Exchange Data with External** function if the previous exchange function returned data. If this parameter is set to **No**, the e*Way will always wait the number of seconds specified by **Exchange Data Interval** between invocations of the **Exchange Data with External** function. The default is **No**.

See **"Exchange Data with External Function" on page 56** for more information.

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

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

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

- Exchange Data With External Function
- Acknowledgment Function

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

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an acknowledgment to the external system (using the Acknowledgment function) and whether the connection to the external system is active. If no acknowledgment is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External** function. Thereafter, the **Exchange Data with External** function will be called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

See **"Exchange Data with External Function" on page 56**, **"Exchange Data Interval" on page 44**, and **"Stop Exchange Data Schedule" on page 45** for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

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

Down Timeout

Description

Specifies the number of seconds that the e*Way will wait between calls to the **External Connection Establishment** function. See **"External Connection Establishment Function" on page 56** for more information.

Required Values

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

Up Timeout

Description

Specifies the number of seconds the e*Way will wait between calls to the **External Connection Verification** function. See **"External Connection Verification Function" on page 57** for more information.

Required Values

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

Resend Timeout

Description

Specifies the number of seconds the e*Way will wait between attempts to resend a message (Event) to the external system, after receiving an error message from the external system.

Required Values

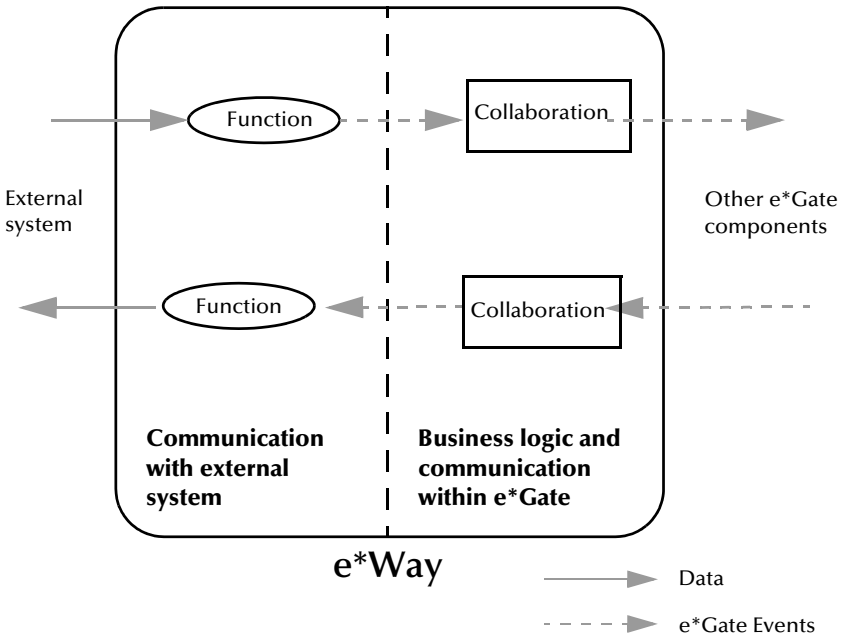
An integer between 1 and 86,400. The default is 10.

4.1.3 DLL Configuration

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

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 10) handles communication with the external system; the other half manages the Collaborations that process data and subscribe or publish to other e*Gate components.

Figure 10 e*Way Internal Architecture



The "communications half" of the e*Way uses the DLL functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events" and send those Events to Collaborations, and manage the connection between the e*Way and the external system. The **DLL Configuration** options discussed in this section control the environment and define the functions used to perform these basic e*Way operations. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as **notepad**, or UNIX **vi**).

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

Operational Details

The functions in the "communications half" of the e*Way fall into the following groups:

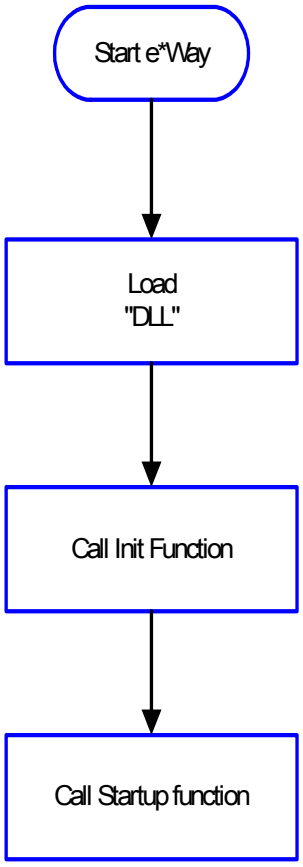
Type of Operation	Name
Initialization	**Startup Function** on page 55
Connection	**External Connection Establishment Function** on page 56 **External Connection Verification Function** on page 57 **External Connection Shutdown Function** on page 57
Schedule-driven data exchange	**Exchange Data with External Function** on page 56 **Acknowledgment Function** on page 58 **Negative Acknowledgment Function** on page 32
Shutdown	**Shutdown Command Notification Function** on page 58
Event-driven data exchange	**Process Outgoing Message Function** on page 55

A series of figures on the next several pages illustrate the interaction and operation of these functions.

Initialization Functions

Figure 11 illustrates how the e*Way executes its initialization functions.

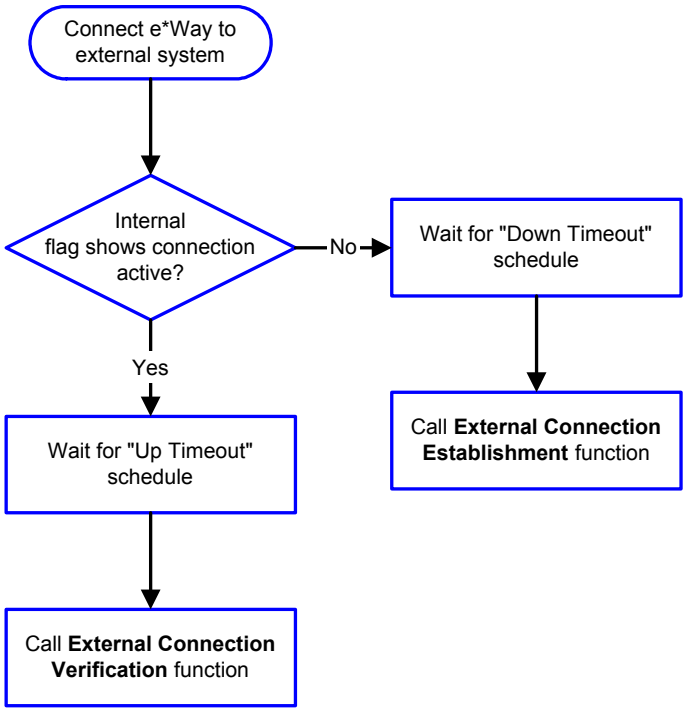
Figure 11 Initialization Functions



Connection Functions

Figure 12 illustrates how the e*Way executes the connection establishment and verification functions.

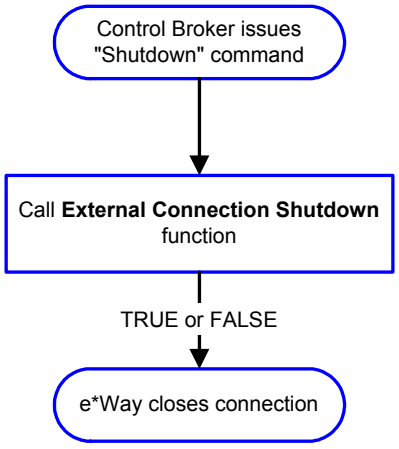
Figure 12 Connection establishment and verification functions



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 14 on page 52** *and* **Figure 16 on page 54** *for examples of how different functions use this flag.*

Figure 13 illustrates how the e*Way executes its "connection shutdown" function.

Figure 13 Connection shutdown function



Schedule-driven Data Exchange Functions

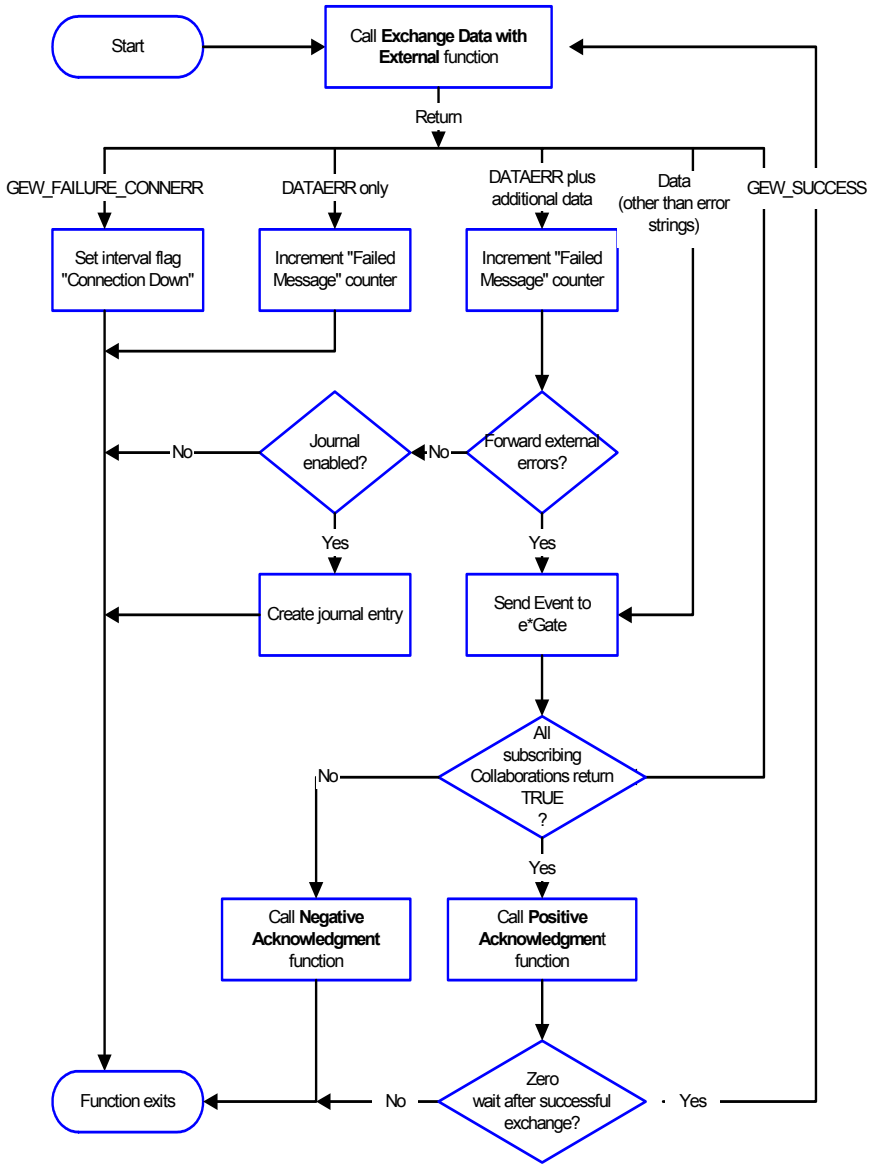
Figure 14 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

"Start" can occur in any of the following ways:

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

After the function exits, the e*Way waits for the next "start schedule" time or command.

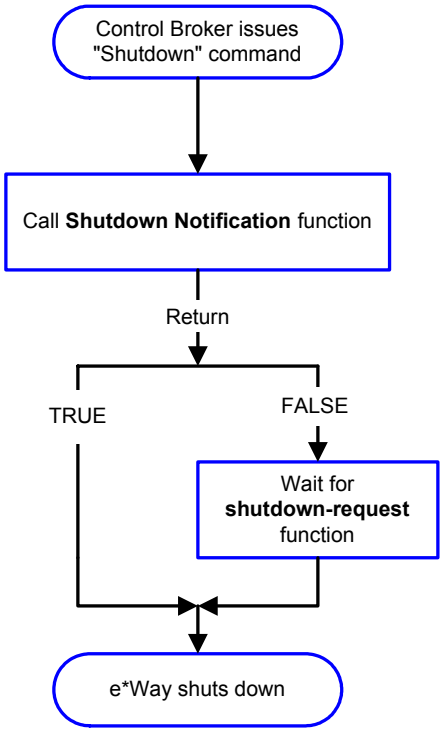
Figure 14 Schedule-driven data exchange functions



Shutdown Functions

Figure 15 illustrates how the e*Way implements the shutdown request function.

Figure 15 Shutdown functions



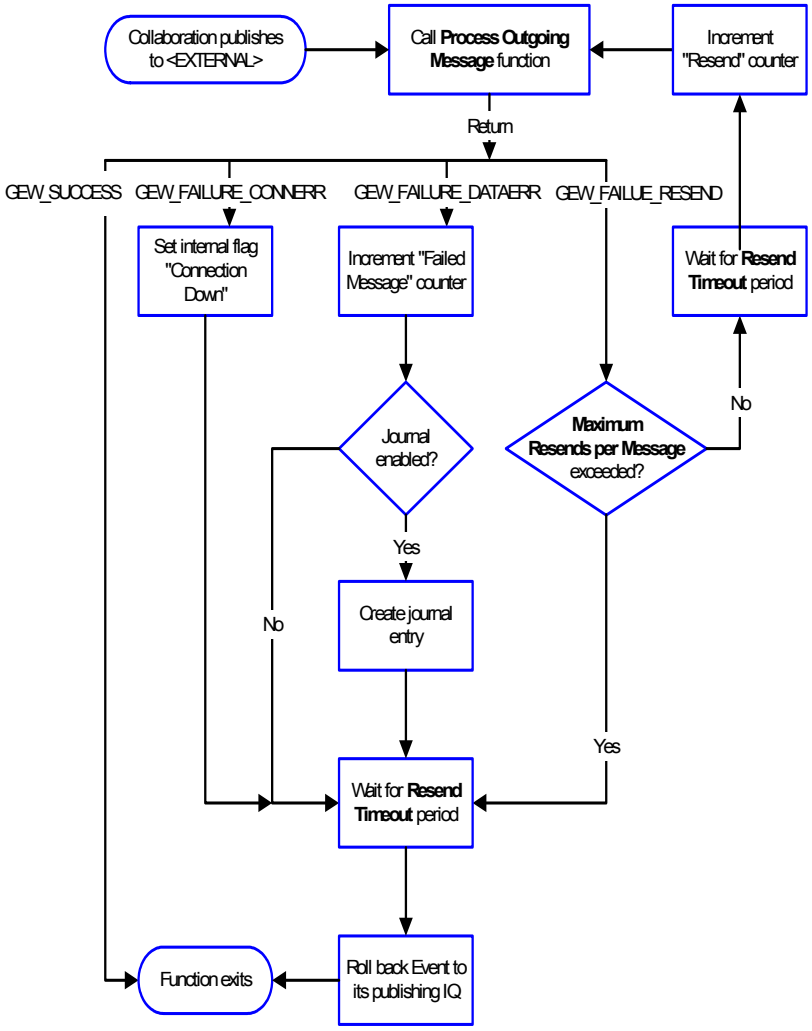
Event-driven Data Exchange Functions

Figure 16 on the next page illustrates event-driven data-exchange using the **Process Outgoing Message Function**.

Every two minutes, the e*Way 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 16 Event-driven data-exchange functions



Dynamic Load Library File

Description

Specifies the DLL that will implement the configured functions within this section.

Required Values

A pathname, or a series of paths separated by semicolons. This parameter is optional and may be left blank.

Additional information

The default load paths are determined by the “bin” and “Shared Data” settings in the .egate.store file. See the *e*Gate Integrator System Administration and Operations Guide* for more information about this file.

Startup Function

Description

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

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function. This parameter is optional and may be left blank.

Additional information

The function accepts no input, and must return a string.

The string "FALSE" indicates that the function failed; "TRUE" indicates success.

This function will be called after the e*Way loads the specified "Environment Initialization file" and any files within the specified **Auxiliary Directories**.

Process Outgoing Message Function

Description

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

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function. ***You may not leave this field blank.***

Additional Information

The function requires a non-null string as input (the outgoing Event to be sent) and must return a string.

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Schema Designer). The function returns one of the following (see **Figure 16 on page 54** for more details):

- TRUE + GEW_SUCCESS: Indicates that the Event was published successfully to the external system.
- FALSE + GEW_FAILURE_RESEND: Indicates that the Event should be resent.
- FALSE + GEW_FAILURE_CONNERR: Indicates that there is a problem communicating with the external system.
- FALSE + GEW_FAILURE_DATAERR: Indicates that there is a problem with the message (Event) data itself.
- If a string other than the above mentioned is returned, the e*Way will create an entry in the log file indicating that an attempt has been made to access an unsupported function.

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. See* **event-send-to-egate** *on page 41* *for more information.*

Exchange Data with External Function

Description

Specifies a function that initiates the transmission of data from the external system to the e*Gate system and forwards that data as an inbound Event to one or more e*Gate Collaborations. This function is called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven).

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function. This parameter is optional and may be left blank.

Additional Information

The function accepts no input and must return a string (see **Figure 14 on page 52** for more details):

- TRUE + GEW_SUCCESS: Indicates that the Event was published successfully to the external system.
- FALSE + GEW_FAILURE_RESEND: Indicates that the Event should be resent.
- FALSE + GEW_FAILURE_CONNERR: Indicates that there is a problem communicating with the external system.
- FALSE + GEW_FAILURE_DATAERR: Indicates that there is a problem with the message (Event) data itself.
- Any other string: The contents of the string are packaged as an inbound Event. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

This function is initially triggered by the **Start Data Exchange** schedule or manually by the function **start-schedule**. After the function has returned true and the data received by this function has been acknowledged (by the **Acknowledgment Function**), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter. If this parameter is set to **Yes**, the e*Way will immediately call the **Exchange Data with External** function again; otherwise, the e*Way will not call the function until the next scheduled "start exchange" time.

External Connection Establishment Function

Description

Specifies a function that the e*Way will call when it has determined that the connection to the external system is down.

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function. ***This field cannot be left blank.***

Additional Information

The function accepts no input and must return a string:

- "TRUE" : Indicates that the connection was established successfully.
- "FALSE": Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Down Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Verification** function (see below) is called when the e*Way has determined that its connection to the external system is up.

External Connection Verification Function

Description

Specifies a function that the e*Way will call when its internal variables show that the connection to the external system is up.

Required Values

The name of a function. This function is optional; if no **External Connection Verification** function is specified, the e*Way will execute the **External Connection Establishment** function in its place.

Additional Information

The function accepts no input and must return a string:

- "TRUE": Indicates that the connection was established successfully.
- "FALSE": Indicates that the attempt to establish the connection failed.

This function is executed according to the interval specified within the **Up Timeout** parameter, and is *only* called according to this schedule.

The **External Connection Establishment** function (see above) is called when the e*Way has determined that its connection to the external system is down.

External Connection Shutdown Function

Description

Specifies a function that the e*Way will call to shut down the connection to the external system.

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function. This parameter is optional.

Acknowledgment Function

Description

Specifies a function that the e*Way will call when the external Event has been processed.

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function. This parameter is required if the **Exchange Data with External** function is defined.

Additional Information

The function requires a non-null string as input (the Event to be sent to the external system) and must return a string:

- TRUE: The function completed execution successfully.
- FALSE : Indicates a problem with the connection to the external system. When the connection is re-established, the Acknowledgment function will be called again, with the same input data.

After the **Exchange Data with External** function returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the Positive Acknowledgment function (otherwise, the e*Way executes the Negative Acknowledgment function).

Shutdown Command Notification Function

Description

Specifies a function that will be called when the e*Way receives a "shut down" command from the Control Broker. This parameter is optional.

Required Values

The name of a function, or the name of a file (optionally including path information) containing a function.

Additional Information

The function accepts a string as input and must return a string:

- TRUE: Indicates that the shutdown can occur immediately.

If there are no external configuration requirements, do not omit this section; instead, insert the following paragraph:

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

Chapter 5

External Interface

This chapter provides the entire C Generic e*Way header file and describes how the header file is used with the external interface object. The overview discusses important concepts and conventions and subsequent sections discuss each type definition in detail.

5.1 Overview

The C Generic e*Way allows you to create a dynamic-link library (DLL) (for Windows 2000), or a shared library (for UNIX), that contains an external interface object. This object holds the types and methods that can be passed between the external application and the C Generic e*Way.

It is important to understand that the C Generic e*Way expects the functions written in your DLL or shared library to be C exports. If you write your DLL or shared library in C++, you MUST declare all functions as extern "C". If you do not export the functions as extern "C", the compiler will mangle the function names and the e*Way will not be able to find your function. Exporting the functions as extern "C" prevents this name mangling.

Note: *For Windows users only: if your functions are written in C and you are using the* ***.cxx*** *extension, all of your functions must be wrapped as extern "C".*

It is also important to adhere to certain calling conventions depending on the platform where you are creating your extension. As an example, see APIDEF in the function pointer prototypes in the **"C Generic e*Way Header File"**. Using APIDEF as an example, Win32 would expect your functions to be exported with the WINAPI calling convention. However, in UNIX this would be defined as white space.

5.2 C Generic e*Way Header File

The external interface object defines a structure that contains a location where the user can place data for the object as well as functions that implement the interface. The functions are defined in the section **"Method Prototypes" on page 68**.

```
#ifndef EWGENERICDLLFUNCS_H
#define EWGENERICDLLFUNCS_H
```

```
#ifdef __cplusplus
extern "C"
{
#endif


typedef void                *HGENEWAYDLL;

typedef enum
{
    GEW_SUCCESS         = 0,
    GEW_FAILURE_CONNERR = 1,
    GEW_FAILURE_DATAERR = 2,
    GEW_FAILURE_RESEND  = 3

} eGEWState_;


//------------------------------------------------------------------
//  PFNEWGENDLL_STARTUP
//  ------------------------------------------------------------------
//
//  Purpose:
//
//      initializes the external dll.
//
//  ------------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.  this is the handle to be used with the //
"AddUserData" and "GetUserData" helper function.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//           occurred.
//
//------------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_STARTUP)(IN HGENEWAYDLL hExt,
                              IN DWORD dwFlags,
                              IN OUT OPTIONAL void *pvReserved);

//------------------------------------------------------------------
//  PFNEWGENDLL_SHUTDOWN
//  ------------------------------------------------------------------
//
//  Purpose:
//
//      called just before the e*Way shuts down.
//
//  ------------------------------------------------------------------
//
//  Parameters:
//
```

```
//  hExt:   state handle.  any user data that has been associated
//       with it must be freed at this time.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//       occured.
//
//-------------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_SHUTDOWN)(IN HGENEWAYDLL hExt,
                               IN DWORD dwFlags,
                               IN OUT OPTIONAL void *pvReserved);

//-------------------------------------------------------------------
//  PFNEWGENDLL_CONNECTIONESTABLISH
//  -----------------------------------------------------------------
//
//  Purpose:
//
//       this entry point is called when the e*Way comes up and if
//       the e*Way's state is "down".
//
//       if the return is FALSE, the e*Way's state becomes "down",
//       if TRUE, it becomes "up".
//
//  -----------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//       occured.
//
//-------------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_CONNECTIONESTABLISH)(IN HGENEWAYDLL hExt,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved);

//-------------------------------------------------------------------
//  PFNEWGENDLL_CONNECTIONVERIFY
//  -----------------------------------------------------------------
//
//  Purpose:
//
//       this entry point is called on the schedule and when the
//       e*Way's state is "up".
//
```

```
//      if the return is FALSE, the e*Way's state becomes "down",
//      if TRUE, it becomes "up".
//
//  ------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//      occured.
//
//------------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_CONNECTIONVERIFY)(IN HGENEWAYDLL hExt,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved);

//------------------------------------------------------------------
//  PFNEWGENDLL_CONNECTIONSHUTDOWN
//  ------------------------------------------------------------
//
//  Purpose:
//
//      this entry point is called when the e*Way needs to shutdown
//      all external connections.
//
//  ------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//      occured.
//
//------------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_CONNECTIONSHUTDOWN)(IN HGENEWAYDLL hExt,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved);


//------------------------------------------------------------------
//  PFNEWGENDLL_PROCESSOUTGOINGEVENT
//  ------------------------------------------------------------
//
//  Purpose:
```

```
//
//      this entry point is called when an event comes from e*Gate
//      destined for the external system.
//
//  ------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  pcEvent:
//          read-only pointer to the event data.
//
//  peState:
//          status enumeration returned to the e*Way to indicate a
//           state change.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//      occured.
//
//------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_PROCESSOUTGOINGEVENT)(IN HGENEWAYDLL hExt,
                      IN PCSTC_BLOB pcEvent,
                      OUT eGEWState_ *peState,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved);


//------------------------------------------------------------------
//  PFNEWGENDLL_GETEXTERNALEVENT
//  ------------------------------------------------------------
//
//  Purpose:
//
//      this entry point is called on one of the e*Way's exchange
//      data schedules to "poll" the external for data to send to
//      e*Gate.
//
//  ------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  pEvent:
//          an initialized STC_BLOB structure is passed in and if
//           there is external data available, the callee should
//           allocate the data member of this structure and fill
//           in the size for the return. the ACK or NAK entry point
//           will get called with the same pEvent, at this point,
//           the data can be freed.
//
//  peState:
//          status enumeration returned to the e*Way to indicate a
//           state change.
```

```
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if the open was successful, FALSE if an error
//       occured.
//
//------------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_GETEXTERNALEVENT)(IN HGENEWAYDLL hExt,
                        OUT STC_BLOB *pEvent,
                        OUT eGEWState_ *peState,
                        IN DWORD dwFlags,
                        IN OUT OPTIONAL void *pvReserved);


//------------------------------------------------------------------
//  PFNEWGENDLL_EXTERNALEVENTACKNOWLEDGEMENT
//  ----------------------------------------------------------------
//
//  Purpose:
//
//      this entry point is called when the external event has been
//      processed by e*Gate.  The "fSuccess" parameter indicates
//      whether e*Gate processing was successful or not - FALSE
//      indicating "not".
//
//  ----------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  fSuccess:
//          TRUE if the external event was successfully processed
//           by e*Gate, FALSE if an error occurred.
//
//  pEvent:
//          an STC_BLOB structure whose members were previously
//           filled in by the GetExternalEvent method.  It is this
//           entry point's responsibility to free any memory that
//           was allocated for the event during the
//           GetExternalEvent call.
//
//  pcExtraCollabReturnData:
//          an OPTIONAL read-only pointer to an STC_BLOB structure
//       that contains any return structure from the collaboration.
//       Do not try to free this data!
//
//  peState:
//          status enumeration returned to the e*Way to indicate a
//           state change.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
```

```
//  return: TRUE if the open was successful, FALSE if an error
//  occurred.
//
//-----------------------------------------------------------------

typedef
BOOL
(APIEXP *PFNEWGENDLL_EXTERNALEVENTACKNOWLEDGEMENT)
                (IN HGENEWAYDLL hExt,
                 IN BOOL fSuccess,
                 IN STC_BLOB *pEvent,
                 IN PCSTC_BLOB pcExtraCollabReturnData,
                 OUT eGEWState_ *peState,
                 IN DWORD dwFlags,
                 IN OUT OPTIONAL void *pvReserved);



//
//  the following helper functions are exposed from
//      stc_stcapis.lib/dll
//

//-----------------------------------------------------------------
//  GenericEWayHelperSetUserData
//  ---------------------------------------------------------------
//
//  Purpose:
//
//      the extension dll should call this function in thier init
//      function to add a reference to thier data to the
//      HGENEWAYDLL handle.
//
//      Use the GetUserData function to retrieve it.  There is only
//      one "slot" for user data.
//
//      On the shutdown function, the dll should call the
//      GetUserData and free the contents.
//
//  ---------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  cbData: number of bytes pointed to by pbData.
//
//  pvData: pointer to the data to be stored in hExt.  This memory
//      is set by reference only and is not duplicated in hExt.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if successful, FALSE if an error occured.
//
//-----------------------------------------------------------------

extern
DLLEXP
BOOL
APIDEF
```

```
GenericEWayHelperSetUserData(IN HGENEWAYDLL hExt,
                             IN DWORD cbData,
                             IN void *pvData,
                             IN DWORD dwFlags,
                             IN OUT OPTIONAL void *pvReserved);



//--------------------------------------------------------------
//  GenericEWayHelperGetUserData
//  --------------------------------------------------------------
//
//  Purpose:
//
//      the extension dll should call this function to access
//      previously stored application data.  this is done by using
//      the AddUserData function.
//
//      the address returned is the same memory address that was
//      originally stored with the AddUserData function.  in other
//      words, freeing this pointer will free the originally
//      allocated memory.
//
//  --------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  pcbData:
//          number of bytes pointed to by ppbData.
//
//  ppvData:
//          pointer to a pointer to the data originally stored in
//      hExt.
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if successful, FALSE if an error occurred.
//
//--------------------------------------------------------------

extern
DLLEXP
BOOL
APIDEF
GenericEWayHelperGetUserData(IN HGENEWAYDLL hExt,
                             OUT DWORD *pcbData,
                             OUT void **ppvData,
                             IN DWORD dwFlags,
                             IN OUT OPTIONAL void *pvReserved);


//--------------------------------------------------------------
//  GenericEWayHelperGetConfigVariable
//  --------------------------------------------------------------
//
//  Purpose:
//
```

```
//      to allow the extension DLL to retrieve configuration
//      parameter that the e*Way extracted from its configuration
//      file.
//
//  ------------------------------------------------------------
//
//  Parameters:
//
//  hExt:   state handle.
//
//  pcszSectionName:
//          configuration section name.
//
//  pcszItemName:
//          configuration item name
//
//  pcbReturnValue:
//          size in bytes of the string returned.  [IN] should be //
set to the maximum length that can be returned in pszValue.
//
//  pszReturnValue:
//          optional buffer to recieve the value string.  if this
//      parameter is NULL or is not large enough to hold the value,
//      this function will return FALSE and the GETLASTERROR()
//      will be set to GENERROR_INSUFFICIENT_DATA
//
//  dwFlags:
//          bit flags.  Reserved for future use.
//
//  pvReserved:
//          this param is reserved for future use.
//
//  return: TRUE if successful, FALSE if an error occured.
//
//------------------------------------------------------------

extern
DLLEXP
BOOL
APIDEF
GenericEWayHelperGetConfigVariable(IN HGENEWAYDLL hExt,
                        IN const char *pcszSectionName,
                        IN const char *pcszItemName,
                        IN OUT DWORD *pcbReturnValue,
                        IN OUT OPTIONAL char *pszReturnValue,
                        IN DWORD dwFlags,
                        IN OUT OPTIONAL void *pvReserved);


#ifdef __cplusplus
}
#endif

#endif // EWGENERICDLLFUNCS_H
```

5.2.1 Type Definitions List

eGEWState_

Structure

```
typedef enum
```

```
{
    GEW_SUCCESS          = 0,
    GEW_FAILURE_CONNERR = 1,
    GEW_FAILURE_DATAERR = 2,
    GEW_FAILURE_RESEND   = 3

} eGEWState_;
```

Description

These are the types of return values for state queries.

GEW_SUCCESS	Success
GEW_FAILURE_CONNERR	Connection Error
GEW_FAILURE_DATAERR	Data Error
GEW_FAILURE_RESEND	Resend

5.3 Method Prototypes

The file **ewgenericdllfuncs.h** defines the following function prototypes:

PFNEWGENDLL_STARTUP

Syntax

PFNEWGENDLL_STARTUP (hExt, dwFlags, pvReserved)

Description

PFNEWGENDLL_STARTUP initializes the external dll.

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the "AddUserData" and "GetUserData" helper functions.
dwFlags	DWORD	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_SHUTDOWN

Syntax

PFNEWGENDLL_SHUTDOWN (hExt, dwFlags, pvReserved)

Description

PFNEWGENDLL_SHUTDOWN is called to request the e*Way shutdown the external connection.

Parameters

Name	Type	Description
hExt	state handle	This is the handle previously associated with user data, that must be freed at this time.
dwFlags	DWORD	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_CONNECTIONESTABLISH

Syntax

PFNEWGENDLL_CONNECTIONESTABLISH (hExt, dwFlags, pvReserved)

Description

PFNEWGENDLL_CONNECTIONESTABLISH is called when the e*Way state comes "up" and if the e*Way's state is "down". If the return value is FALSE, the e*Way's state is defined as "down", if TRUE, the state is defined as "up".

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	DWORD	Bit flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_VERIFY

Syntax

PFNEWGENDLL_VERIFY (hExt, dwFlags, pvReserved)

Description

PFNEWGENDLL_VERIFY is called on the schedule and when the e*Way's state is "up". If the return is FALSE, the e*Way's state is defined as "down", if TRUE, the state is defined as"up".

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	bit flags	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_CONNECTIONSHUTDOWN

Syntax

PFNEWGENDLL_CONNECTIONSHUTDOWN (hExt, dwFlags, pvReserved)

Description

PFNEWGENDLL_CONNECTIONSHUTDOWN is called by the e*Way to shutdown all external connections.

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	bit flags	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_PROCESSOUTGOINGEVENT

Syntax

PFNEWGENDLL_PROCESSOUTGOINGEVENT (hExt, pcEvent, peState, dwFlags, pvReserved)

Description

PFNEWGENDLL_PROCESSOUTGOINGEVENT is called when an Event is received from e*Gate, destined for the external system.

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
pcEvent	pointer	A read-only pointer to the Event data.
peState	integer	Status enumeration returned to the e*Way to indicate a state change.
dwFlags	DWORD	Bit flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_GETEXTERNALEVENT

Syntax

PFNEWGENDLL_GETEXTERNALEVENT (hExt, pEvent, peState, dwFlags, pvReserved)

Description

PFNEWGENDLL_STARTUP is called by one of the e*Way's exchange data schedules when the e*Way "polls" the external for data to send to e*Gate.

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
pEvent	Blob	An initialized STC_BLOB structure is passed in, if there is external data available, the callee should allocate the data member of this structure and fill in the size for the return. The ACK or NAK entry point will get called with the same pEvent. At this point, the data can be freed.
peState	integer	Status enumeration returned to the e*Way to indicate a state change.
dwFlags	DWORD	Bit flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

PFNEWGENDLL_EXTERNALEVENTACKNOWLEDGEMENT

Syntax

PFNEWGENDLL_EXTERNALEVENTACKNOWLEDGEMENT (hExt, fSuccess, pEvent, pcExtraCollabReturnData, peState, dwFlags, pvReserved)

Description

PFNEWGENDLL_EXTERNALEVENTACKNOWLEDGEMENT is called when the external Event has been processed by e*Gate. The fSuccess parameter indicates whether e*Gate processing was successful or not - FALSE indicating "not successful".

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
fSuccess	TRUE or FALSE	Indicates whether the e*Gate processed the external Event successfully.
pEvent	BLOB	An STC_BLOB structure whose members were previously filled in by **GetExternalEvent** method. This entry point frees any memory allocated for the Event during the **GetExternalEvent** call.
pcExtraCollabReturnData	pointer	An optional read-only point to an STC_BLOB structure that contains any return structure from the collaboration. Do not try to free this data.
peState	integer	Status enumeration returned to the e*Way to indicate a state change.
dwFlags	bit flags	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

5.4 Helper Functions

The following helper functions are exposed from **stc_stcapis.lib/dll**:

GenericEWayHelperSetUserData on page 73

GenericEWayHelperGetUserData on page 74

GenericEWayHelperGetConfigVariable on page 75

GenericEWayHelperSetUserData

Syntax

GenericEWayHelperSetUserData (hExt, cbData, pvData, dwFlags, pvReserved)

Description

GenericEWayHelperSetUserData is called by the extension dll in the corresponding init function to add a reference to its data to the **HGENEWAYDLL** handle. Once the handle has been stored, use the **GetUserData** function to retrieve it. There is only one "slot" for user data. In the shutdown function, the dll should call the **GetUserData** pointer and free the contents.

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
cbData	integer	The number of bytes pointed to by pbData
pvData	pointer	A pointer to the data to be stored in hExt. This memory is set by reference only and is not duplicated in hExt.
dwFlags	DWORD	Bit Flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

GenericEWayHelperGetUserData

Syntax

GenericEWayHelperGetUserData (hExt, pcbData, ppvData, dwFlags, pvReserved)

Description

GenericEWayHelperGetUserData is called by the extension dll to access previously stored application data. The data was stored by using the **SetUserData** function. The address returned is the same memory address that was originally allocated memory by **SetUserData**.

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
pcbData	integer	The number of bytes pointed to by ppbData
ppvData	pointer	A pointer to a pointer to the data originally stored in hExt.

Name	Type	Description
dwFlags	DWORD	Bit Flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

GenericEWayHelperGetConfigVariable

Syntax

```
GenericEWayHelperGetConfigVariable (hExt, pcszSectionName,
pcszItemName, pcbReturnValue,pszReturnValue, dwFlags, pvReserved)
```

Description

GenericEWayHelperGetConfigVariable allows the extension dll to retrieve the configuration parameters that the e*Way extracted from the corresponding configuration file.

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
pcszSectionName	pointer control string zero delimited.	The configuration section name.
pcszItemName	pointer control string zero delimited	A pointer to a pointer to the data originally stored in **hExt**.
pcbReturnValue	integer	The size in bytes of the string returned. It should be set to the maximum length that can be returned in **pszValue**.
pszReturnValue	zero delimited pointer	An optional buffer to receive the value string. If this parameter is NULL or is not large enough to hold the value, this functions will return FALSE and the **GETLASTERROR()** will be set to **GENERROR_INSUFFICIENT_DATA**.
dwFlags	DWORD	Bit Flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

5.5 Sample Template Source Code

The following sample source code is a template that provides an example of how to use the C Generic e*Way's headers, types, methods, and functions to provide access to an external application.

The header files that are referenced as include files can be copied from the SDK folder on the root level of the installation CD.

A detailed description of the source code (ewgenericdll) follows the sample.

```
#ifndef EWGENDLLEXTFILE_H
#define EWGENDLLEXTFILE_H

#include    "gendefs.h"
#include    "stcapis.h"
#include    "ewgenericdllfuncs.h"

#ifdef __cplusplus
extern "C"
{
#endif

typedef struct GENDLLEXT_FILEEXT_DATA_
{
    DWORD           cbStruct;   // set to
sizeof(GENDLLEXT_FILEEXT_DATA)

    DWORD           cRecvEvents;

    char            szToEgateFile[MAX_PATH];

    char            szFromEgateFile[MAX_PATH];
    FILE            *pFromEgateFile;

} GENDLLEXT_FILEEXT_DATA;

extern
DLLEXP
BOOL
APIDEF
FileExt_Startup(IN HGENEWAYDLL hExt,
                IN DWORD dwFlags,
                IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_Shutdown(IN HGENEWAYDLL hExt,
                 IN DWORD dwFlags,
                 IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_ConnEstablish(IN HGENEWAYDLL hExt,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved);

extern
```

```
DLLEXP
BOOL
APIDEF
FileExt_ConnVerify(IN HGENEWAYDLL hExt,
                   IN DWORD dwFlags,
                   IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_ConnShutdown(IN HGENEWAYDLL hExt,
                     IN DWORD dwFlags,
                     IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_SendExternalEvent(IN HGENEWAYDLL hExt,
                          IN PCSTC_BLOB pcEvent,
                          OUT eGEWState_ *peState,
                          IN DWORD dwFlags,
                          IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_GetExternalEvent(IN HGENEWAYDLL hExt,
                         OUT STC_BLOB *pEvent,
                         OUT eGEWState_ *peState,
                         IN DWORD dwFlags,
                         IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_EventAckknowledgement(IN HGENEWAYDLL hExt,
                              IN BOOL fSuccess,
                              IN STC_BLOB *pEvent,
                              IN PCSTC_BLOB pcExtraCollabReturnData,
                              OUT eGEWState_ *peState,
                              IN DWORD dwFlags,
                              IN OUT OPTIONAL void *pvReserved);

#ifdef __cplusplus
}
#endif

#endif // EWGENDLLEXTFILE_H
```

5.6 Sample Source Code Description

This section is a description of the ewgendllext.h Source Code Sample that creates an application-specific dynamic link library (dll) or shard library (.so) for extending the e*Way functionality.

These are the standard header files needed by this dynamic or shared library:

```
#include    "gendefs.h"
#include    "stcapis.h"
```

This is the header file that defines the structures, parameters, methods and functions used by this e*Way:

```
#include    "ewgenericdllfuncs.h"
```

This indicates that this is a C++ header file:

```
#ifdef __cplusplus
extern "C"
{
#endif
```

This is the Type Definition GENDLLEXT_FILEEXT_DATA

```
typedef struct GENDLLEXT_FILEEXT_DATA_
{
    DWORD           cbStruct;   // set to
sizeof(GENDLLEXT_FILEEXT_DATA)

    DWORD           cRecvEvents;

    char            szToEgateFile[MAX_PATH];

    char            szFromEgateFile[MAX_PATH];
    FILE            *pFromEgateFile;

} GENDLLEXT_FILEEXT_DATA;
```

The following are the external API definitions:

```
extern
DLLEXP
BOOL
APIDEF
FileExt_Startup(IN HGENEWAYDLL hExt,
                IN DWORD dwFlags,
                IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_Shutdown(IN HGENEWAYDLL hExt,
                 IN DWORD dwFlags,
                 IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_ConnEstablish(IN HGENEWAYDLL hExt,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_ConnVerify(IN HGENEWAYDLL hExt,
                   IN DWORD dwFlags,
                   IN OUT OPTIONAL void *pvReserved);
```

```
extern
DLLEXP
BOOL
APIDEF
FileExt_ConnShutdown(IN HGENEWAYDLL hExt,
                     IN DWORD dwFlags,
                     IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_SendExternalEvent(IN HGENEWAYDLL hExt,
                          IN PCSTC_BLOB pcEvent,
                          OUT eGEWState_ *peState,
                          IN DWORD dwFlags,
                          IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_GetExternalEvent(IN HGENEWAYDLL hExt,
                         OUT STC_BLOB *pEvent,
                         OUT eGEWState_ *peState,
                         IN DWORD dwFlags,
                         IN OUT OPTIONAL void *pvReserved);

extern
DLLEXP
BOOL
APIDEF
FileExt_EventAckknowledgement(IN HGENEWAYDLL hExt,
                              IN BOOL fSuccess,
                              IN STC_BLOB *pEvent,
                              IN PCSTC_BLOB pcExtraCollabReturnData,
                              OUT eGEWState_ *peState,
                              IN DWORD dwFlags,
                              IN OUT OPTIONAL void *pvReserved);
```

Chapter 6

Suggested Method Implementation

This chapter includes information pertinent to implementing the C Generic e*Way.

6.0.1 Considerations

The JCS can not be run from the collaboration rule associated with the e*Way's collaboration. It is however, possible to run the JCS from another e*Way (or work slice, such as a file e*Way) publish to a queue and pass the translated information to the C Generic e*Way via a queue.

6.1 Sample C APIs

The following sample methods demonstrate the suggested usage for each of the eight APIs present. Any additional APIs created for the application-specific C Generic e*Way, should use these samples as templates.

The naming of the methods within the sample use the following naming conventions:

```
ApplicationName_FunctionName
```

For the basic functions, it is recommended that the developer maintain the function name, while editing/adapting the application-specific name. In the sample, **FileExt** replaced **ApplicationName**.

startup.cxx

FileExt_Startup

Syntax

FileExt_Startup (hExt, dwFlags, *pvReserved)

Description

FileExt_Startup initializes the external dll.

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	DWORD	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_Startup(IN HGENEWAYDLL hExt,
                IN DWORD dwFlags,
                IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;

    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    pExtData = (GENDLLEXT_FILEEXT_DATA *)malloc(sizeof(GENDLLEXT_FILEEXT_DATA));

    if (!(pExtData))
    {
        goto MallocFailed;
    }

    memset(pExtData, 0x00, sizeof(GENDLLEXT_FILEEXT_DATA));

    pExtData->cbStruct = sizeof(GENDLLEXT_FILEEXT_DATA);

    cb = MAX_PATH;

    if (!(GenericEWayHelperGetConfigVariable(hExt,
                                             "File Settings",
                                             "To eGate File Name",
                                             &cb,
                                             pExtData->szToEgateFile,
                                             0,
                                             NULL)))
    {
        TRACE_0(TRACE_CONFIGURATION, TRACE_EVENT_FATAL,
                "unable to get setting: File Settings - To eGate File Name");

        goto ParamError;
    }

    cb = MAX_PATH;

    if (!(GenericEWayHelperGetConfigVariable(hExt,
                                             "File Settings",
                                             "From eGate File Name",
                                             &cb,
                                             pExtData->szFromEgateFile,
                                             0,
                                             NULL)))
    {
        TRACE_0(TRACE_CONFIGURATION, TRACE_EVENT_FATAL,
                "unable to get setting: File Settings - From eGate File Name");

        goto ParamError;
    }

    if (!(GenericEWayHelperSetUserData(hExt,
                                       sizeof(GENDLLEXT_FILEEXT_DATA),
                                       pExtData,
                                       0,
                                       NULL)))
    {
        goto AddUserDataFailed;
    }

    fRet = TRUE;

CommonReturn:
    return(fRet);

ErrorReturn:
    DELETE_MALLOC(pExtData);
    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, ParamError);
TRACE_RETURN_ERROR_SET(TRACE_EWAY, AddUserDataFailed);
TRACE_RETURN(TRACE_EWAY, MallocFailed, GENERROR_MEMORY_ALLOCATION);
}
```

shutdown.cxx

FileExt_Shutdown

Syntax

FileExt_Shutdown (hExt, dwFlags, *pvReserved)

Description

FileExt_Shutdown requests the external to shutdown.

Parameters

Name	Type	Description
hExt	state handle	The handle previously associated with user data, that must be freed at this time.
dwFlags	DWORD	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_Shutdown(IN HGENEWAYDLL hExt,
                 IN DWORD dwFlags,
                 IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;

    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    pExtData = NULL;

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
        goto GetUserDataFailed;
    }

    if (pExtData->pFromEgateFile)
    {
        fclose(pExtData->pFromEgateFile);
        pExtData->pFromEgateFile = NULL;
    }

    GenericEWayHelperSetUserData(hExt, 0, NULL, 0, NULL);

    fRet = TRUE;

CommonReturn:
    DELETE_MALLOC(pExtData);
    return(fRet);

ErrorReturn:
    fRet = FALSE;
```

```
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
}
```

connestablish.cxx

FileExt_ConnEstablish

Syntax

FileExt_ConnEstablish is called when the e*Way state comes "up" and if the e*Way's state is "down". If the return is FALSE, the e*Way's state becomes "down", if TRUE, it becomes "up".

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	DWORD	Bit flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_ConnEstablish(IN HGENEWAYDLL hExt,
                      IN DWORD dwFlags,
                      IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;

    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
        goto GetUserDataFailed;
    }

    if (!(pExtData->pFromEgateFile))
    {
        pExtData->pFromEgateFile = fopen(pExtData->szFromEgateFile, "ab+");

        if (!(pExtData->pFromEgateFile))
        {
            goto fopenFailed;
        }
    }

    fRet = TRUE;
```

```
CommonReturn:
    return(fRet);

ErrorReturn:
    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
TRACE_RETURN_ERROR_SET(TRACE_EWAY, fopenFailed);
}
```

connverify.cxx

FileExt_ConnVerify

Syntax

FileExt_ConnVerify (hExt, dwFlags, *pvReserved)

Description

FileExt_ConnVerify verifies on schedule that the e*Way's state is "up". If the return is FALSE, the e*Way's state becomes "down", if TRUE, the state becomes "up".

Parameters

Name	Type	Description
hExt	state handle	This is the handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	bit flags	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_ConnVerify(IN HGENEWAYDLL hExt,
                   IN DWORD dwFlags,
                   IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;

    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    pExtData = NULL;

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
```

```
        goto GetUserDataFailed;
    }

    if (!(pExtData->pFromEgateFile))
    {
        goto NotOpen;
    }

    fRet = TRUE;

CommonReturn:
    return(fRet);

ErrorReturn:
    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
TRACE_RETURN(TRACE_EWAY, NotOpen, GENERROR_OPEN);
}
```

connshutdown.cxx

FileExt_ConnShutdown

Syntax

FileExt_ConnShutdown (hExt, dwFlags, *pvReserved)

Description

FileExt_ConnShutdown is called to request the e*Way shutdown the external connection.

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
dwFlags	bit flags	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_ConnShutdown(IN HGENEWAYDLL hExt,
                     IN DWORD dwFlags,
                     IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;
```

```
    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
        goto GetUserDataFailed;
    }

    if (pExtData->pFromEgateFile)
    {
        fclose(pExtData->pFromEgateFile);
        pExtData->pFromEgateFile = NULL;
    }

    fRet = TRUE;

CommonReturn:
    return(fRet);

ErrorReturn:
    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
}
```

send.cxx

FileExt_SendExternalEvent

Syntax

```
FileExt_SendExternalEvent (hExt, pcEvent, *peState, dwFlags,
*pvReserved)
```

Description

FileExt_SendExternalEvent is called when an Event is received from e*Gate, destined for the external system.

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
pcEvent	pointer	A read-only pointer to the Event data.

Name	Type	Description
peState	integer	Status enumeration returned to the e*Way to indicate a state change.
dwFlags	DWORD	Bit flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_SendExternalEvent(IN HGENEWAYDLL hExt,
                          IN PCSTC_BLOB pcEvent,
                          OUT eGEWState_ *peState,
                          IN DWORD dwFlags,
                          IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;

    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    pExtData = NULL;

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
        goto GetUserDataFailed;
    }

    if (!(pExtData->pFromEgateFile))
    {
        *peState = GEW_FAILURE_CONNERR;

        goto NotOpen;
    }

    if (fwrite(pcEvent->pbData,
               sizeof(BYTE),
               pcEvent->cbData,
               pExtData->pFromEgateFile) != pcEvent->cbData)
    {
        *peState = GEW_FAILURE_CONNERR;

        fclose(pExtData->pFromEgateFile);
        pExtData->pFromEgateFile = NULL;
    }

    fRet = TRUE;
```

```
CommonReturn:
    return(fRet);

ErrorReturn:
    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
TRACE_RETURN(TRACE_EWAY, NotOpen, GENERROR_OPEN);
}
```

recv.cxx

FileExt_GetExternalEvent

Syntax

FileExt_GetExternalEvent (hExt, *pEvent, *peState, dwFlags, *pvReserved)

Description

FileExt_GetExternalEvent is called by one of the e*Way's exchange data schedules when the e*Way "polls" the external for data to send to e*Gate.

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
pEvent	Blob	If there is external data available, the initialized STC_BLOB structure is passed in. The callee should allocate the data member of this structure and fill in the size for the return. The ACK or NAK entry point will get called with the same **pEvent**. At this point, the data can be freed.
peState	integer	Status enumeration returned to the e*Way to indicate a state change.
dwFlags	DWORD	Bit flags. Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; otherwise, returns FALSE.

Example

```
BOOL
APIDEF
FileExt_GetExternalEvent(IN HGENEWAYDLL hExt,
                         OUT STC_BLOB *pEvent,
                         OUT eGEWState_ *peState,
                         IN DWORD dwFlags,
                         IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;
    FILE                    *pFile;
    size_t                  cbRead;
    long                    cbFile;
    char                    sz[MAX_PATH];
    char                    szRename[MAX_PATH];
    char                    *psz;

    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);

    pExtData    = NULL;
    pFile       = NULL;

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
        goto GetUserDataFailed;
    }

    pFile = fopen(pExtData->szToEgateFile, "rb");

    if (!(pFile))
    {
        goto NotReady;
    }

    if (fseek(pFile, 0, SEEK_END) != 0)
    {
        goto fseekFailed;
    }

    cbFile = ftell(pFile);

    if (cbFile == (-1))
    {
        goto ftellFailed;
    }

    rewind(pFile);

    pEvent->cbData = (DWORD)cbFile;
    pEvent->pbData = (BYTE *)malloc(cbFile);

    if (!(pEvent->pbData))
    {
        goto MallocFailed;
    }

    cbRead = fread(pEvent->pbData, sizeof(BYTE), (size_t)pEvent->cbData, pFile);

    if ((cbRead == 0) ||
        (cbRead != (size_t)pEvent->cbData))
    {
        TRACE_4(TRACE_EWAY, TRACE_EVENT_DEBUG,
                "found file, but read failed cbRead=%lu event cbData=%lu.  Error: %s (0x%08X)",
                (ULONG)cbRead,
                (ULONG)pEvent->cbData,
                GenerrorGetErrorText(GETLASTERROR()),
                GETLASTERROR());

        goto freadFailed;
    }

    fclose(pFile);
    pFile = NULL;

    //
```

```
    //  rename file...
    //

    strcpy(sz, pExtData->szToEgateFile);

    psz = strchr(sz, C_DIRSLASH);

    if (psz)
    {
        *psz = 0x00;
        psz++;
    }
    else
    {
        psz = sz;
    }

    pExtData->cRecvEvents++;

    _chdir(sz);

    sprintf(szRename, "%s_%lu.done", psz, (ULONG)pExtData->cRecvEvents);

    if (rename(psz, szRename) == (-1))
    {

        goto RenameFailed;
    }

    fRet = TRUE;

CommonReturn:
    return(fRet);

ErrorReturn:
    pEvent->cbData = 0;
    DELETE_MALLOC(pEvent->pbData);

    if (pFile)
    {
        fclose(pFile);
    }

    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
TRACE_RETURN_ERROR_SET(TRACE_EWAY, RenameFailed);
TRACE_RETURN_ERROR_SET(TRACE_EWAY, freadFailed);
TRACE_RETURN_ERROR_SET(TRACE_EWAY, ftellFailed);
TRACE_RETURN_ERROR_SET(TRACE_EWAY, fseekFailed);
TRACE_RETURN(TRACE_EWAY, NotReady, GENERROR_NOTREADY);
TRACE_RETURN(TRACE_EWAY, MallocFailed, GENERROR_MEMORY_ALLOCATION);
}
```

ack.cxx

FileExt_EventAcknowledgmenet

Syntax

FileExt_EventAcknowledgment (hExt, fSuccess, *pEvent, pcExtraCollabReturnData, *peState, dwFlags, *pvReserved)

Description

FileExt_EventAcknowledgment is called when the external Event has been processed by e*Gate. The **fSuccess** parameter indicates whether e*Gate processed the data successfully or not - FALSE indicating “not successful”.

Parameters

Name	Type	Description
hExt	state handle	The handle to be used with the **SetUserData** and **GetUserData** helper functions.
fSuccess	TRUE or FALSE	A string that indicates whether the e*Gate processed the external Event successfully.
pEvent	BLOB	An STC_BLOB structure whose members were previously filled in by **GetExternalEvent** method. This entry point frees any memory that was allocated for the Event during the **GetExternalEvent** call.
pcExtraCollabReturnData	pointer	An optional read-only point to an STC_BLOB structure that contains any return structure from the collaboration. Do not try to free this data.
peState	integer	Status enumeration returned to the e*Way to indicate a state change.
dwFlags	bit flags	Reserved for future use.
pvReserved	void	Reserved for future use.

Return Values

Bool

Returns TRUE if successful; FALSE if an error occurred.

Example

```
BOOL
APIDEF
FileExt_EventAckknowledgement(IN HGENEWAYDLL hExt,
                              IN BOOL fSuccess,
                              IN STC_BLOB *pEvent,
                              IN PCSTC_BLOB pcExtraCollabReturnData,
                              OUT eGEWState_ *peState,
                              IN DWORD dwFlags,
                              IN OUT OPTIONAL void *pvReserved)
{
    BOOL                    fRet;
    DWORD                   cb;
    GENDLLEXT_FILEEXT_DATA  *pExtData;
```

```
    GNU_USEARG(dwFlags);
    GNU_USEARG(pvReserved);
    GNU_USEARG(pcExtraCollabReturnData);

    pExtData = NULL;

    if (!(GenericEWayHelperGetUserData(hExt,
                                       &cb,
                                       (void **)&pExtData,
                                       0,
                                       NULL)))
    {
        goto GetUserDataFailed;
    }

    if (pExtData->pFromEgateFile)
    {
        fflush(pExtData->pFromEgateFile);
    }

    fRet = TRUE;

CommonReturn:
    DELETE_MALLOC(pEvent->pbData);
    pEvent->cbData = 0;
    return(fRet);

ErrorReturn:
    fRet = FALSE;
    goto CommonReturn;

TRACE_RETURN_ERROR_SET(TRACE_EWAY, GetUserDataFailed);
}
```

Chapter 7

Configuring the e*Way with the Schema Designer

7.1 Implementing a Sample Schema

The instructions in this chapter discuss how to implement the C Generic e*Way using the Schema Designer.

After you have created the extension **dll**, any required functions, and the .def file (if necessary) for the new e*Way, you must do the following:

1. Commit any files you have created to the appropriate directories within a schema.
2. Create an e*Way component within the schema.
3. Configure the e*Way as required.

7.1.1 Step 1: Commit Files to the Schema

Note: *Do not commit files to the **default** schema unless you want those files to be inherited by all new schemas. Even if this is the desired outcome, we recommend that you always commit files to a non-default schema during testing and development of new e*Way components.*

1. Make sure that the files you wish to commit to the e*Gate schema are accessible from the same system as the Schema Designer GUI, either from a local file system or from a mapped network drive (you cannot commit files to the schema using a UNC path).
2. Using the Schema Designer, login into the schema that will support the new e*Way.
3. Pull down the File menu and select **Commit to Sandbox**.
4. The **Select Local File to Commit** dialog appears. Use the file-selection controls to locate the file you want to commit and click **Open**.
5. The **Select Directory for Committed File** dialog appears. Use the directory-selection controls to locate the directory to you want to commit the file and click **Select**. Select the directory according to those shown in Table 6.

Table 6 Schema directories

For a file of this type…	…commit to this directory
.def	/configs/stcewgenericdll
.monk (e*Way functions)	monk_scripts/***eway_name*** (We recommend that you create a separate directory for your custom e*Way scripts.)
.dll	/bin

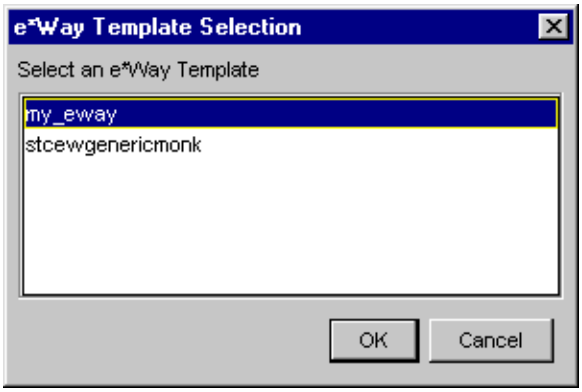
Any ETD (**.ssc**) and Collaboration Rules (**.tsc**) files that you create for this e*Way should be stored in the schema within the **/monk_scripts/common** directory, but you do not need to commit any such files manually if you create them using the e*Gate ETD or Collaboration Rules Editors. If you use another editor to create these files (such as **Notepad**, **Wordpad**, or **vi**), you must commit the files manually.

Note: *Remember that committing files to the Sandbox makes them available for testing. Files must be promoted to the run-time schema before they can be used in the working "production" environment. For more information, see the Team Registry user's guide or the Schema Designer's Help system.*

7.1.2 Step 2: Create an e*Way Component

After all the required files have been committed to the schema, you can create the e*Way component.

1. In the Component editor, create a new e*Way.
2. Display the new e*Way's properties.
3. On the General tab, under **Executable File**, click **Find**.
4. Select the file **stcewgenericdll.exe**.
5. Under **Configuration file**, click **New**.
6. The **e*Way Template Selection** dialog box appears. From the list, select the **.def** file that you created for this e*Way and click **OK**. The name will be listed without the ".def" extension. For example, if you created the file **my_eway.def**, the file will be listed as **my_eway**, as shown below:

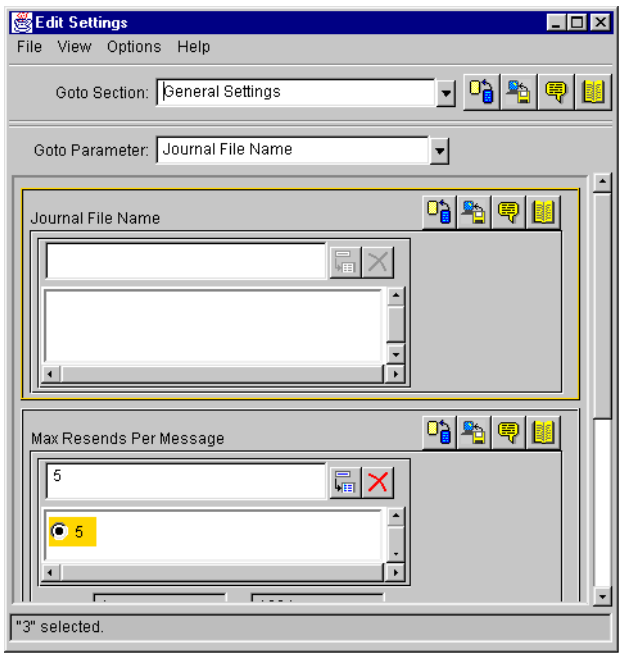


7 The e*Way Editor will launch. You are ready to configure the e*Way; continue with the next section.

7.1.3 Step 3: Configure the e*Way

Once you have selected your e*Way template, you are ready to use the e*Way Editor to configure this e*Way component.

1 If you followed the instructions in the previous two sections, the e*Way Editor has now launched (shown in the figure below).



Use the e*Way Editor to make any configuration changes you require. For more information about configuring e*Ways or how to use the e*Way Editor, see the **e*Gate Integrator User's Guide**.

2 When you have finished making configuration changes, pull down the **File** menu and select **Save**.

3 Enter a name for the configuration file and click **OK**.

4 Exit the e*Way Editor. You will return to the e*Way's property sheet. Click **OK** to close the properties sheet, or continue to make other changes to the e*Way component's properties.

Note: *Once you have installed and configured this e*Way, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the Schema Designer's online Help system.*

7.1.4 Editing a .def File Within a Schema

To edit a .def file that has already been committed to a schema:

1 Launch the Schema Designer and login to the schema containing the **.def** file that you want to edit.

2 Pull down the **File** menu and select **Edit File.**

3 Use the file-selection controls to open the **.def** file. The Notepad editor will launch and open the file you have selected.

4 Save any changes and exit the editor.

5 Commit the edited file back to the schema (the Schema Designer will automatically prompt you to perform this procedure).

See the Schema Designer's online help for more information.

Index

K

L

M

N

P

Q

R

S

T

U

V

W