

SeeBeyond™ eBusiness Integration Suite

e*Way Intelligent Adapter for CICS User's Guide

Release 4.5.2

Monk Version



The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

e*Gate, e*Insight, e*Way, e*Xchange, eBI, eBusiness Web, iBridge, Intelligent Bridge, IQ, SeeBeyond, and the SeeBeyond logo are trademarks and service marks of SeeBeyond Technology Corporation. All other brands or product names are trademarks of their respective companies.

© 1999–2001 by SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20011019182224.

Contents

Chapter 1

| | |
|------------------------------|----------|
| Introduction | 1 |
| Overview | 1 |
| Intended Reader | 1 |
| Components | 2 |
| System Requirements | 2 |
| External System Requirements | 2 |

Chapter 2

| | |
|--|----------|
| Installation | 5 |
| Windows NT and Windows 2000 | 5 |
| Pre-installation | 5 |
| Installation Procedure | 5 |
| UNIX | 6 |
| Pre-installation | 6 |
| Installation Procedure | 6 |
| Files/Directories Created by the Installation | 7 |

Chapter 3

| | |
|--|-----------|
| Configuration | 8 |
| e*Way Configuration Parameters | 8 |
| General Settings | 8 |
| Journal File Name | 9 |
| Max Resends Per Message | 9 |
| Max Failed Messages | 9 |
| Forward External Errors | 9 |
| Communication Setup | 10 |
| Start Exchange Data Schedule | 10 |
| Stop Exchange Data Schedule | 10 |
| Exchange Data Interval | 11 |
| DownTimeout | 11 |
| Up Timeout | 11 |
| Resend Timeout | 12 |
| Zero Wait Between Successful Exchanges | 12 |

| | |
|---|-----------|
| Monk Configuration | 12 |
| Operational Details | 13 |
| How to Specify Function Names or File Names | 21 |
| Additional Path | 21 |
| Auxiliary Library Directories | 21 |
| Monk Environment Initialization File | 22 |
| Startup Function | 22 |
| Process Outgoing Message Function | 23 |
| Exchange Data with External Function | 23 |
| External Connection Establishment Function | 24 |
| External Connection Verification Function | 25 |
| External Connection Shutdown Function | 25 |
| Positive Acknowledgment Function | 26 |
| Negative Acknowledgment Function | 26 |
| Shutdown Command Notification Function | 27 |
| CICS Settings | 27 |
| CICS User | 27 |
| CICS User Password | 27 |
| Environment Configuration | 28 |
| External Configuration Requirements | 28 |

Chapter 4

| | |
|---|-----------|
| Implementation | 29 |
| Implementation Overview | 29 |
| Modes of Operation | 30 |
| Outbound-to-CICS Mode | 30 |
| Inbound-from-CICS Mode | 31 |
| Request/reply Mode | 31 |
| Using the Cobol Copybook Converter | 32 |
| ECI Error Codes | 35 |

Chapter 5

| | |
|-----------------------------|-----------|
| CICS e*Way Functions | 39 |
| Basic Functions | 39 |
| event-send-to-egate | 40 |
| get-logical-name | 41 |
| send-external-down | 42 |
| send-external-up | 43 |
| shutdown-request | 44 |
| start-schedule | 45 |
| stop-schedule | 46 |
| CICS Functions | 46 |
| EciListSystems | 47 |
| ExternalCall | 48 |

Introduction

This chapter includes a brief description of IBM's Customer Information Control System (CICS), an overview of SeeBeyond™ Technology Corporation's (SeeBeyond™) e*Way Intelligent Adapter for CICS, as well as system requirements for using the e*Way.

1.1 Overview

The CICS e*Way is an interface that makes bidirectional calls to CICS transactional programs remotely. CICS is a transaction processor that supports a real-time distributed processing environment and also supports online transaction processing (OLTP).

IBM provides a CICS Client Gateway that has an API (the External Call Interface or ECI) to call CICS transactions on the mainframe. The ECI allows a non-CICS application program to call a CICS program in a CICS server. SeeBeyond's CICS e*Way uses this ECI method to connect to CICS.

The CICS e*Way includes a build tool, the Cobol Copybook Converter. This feature takes as input a Cobol Copybook file and creates e*Gate Event Type Definitions (ETDs) for use within the Monk environment. These Copybook file structures are passed into the CICS environment as the data buffer (COMMAREACOMMAREA).

The CICS e*Way has the following modes of operation:

- Inbound
- Outbound
- Request/reply

For more information, see [“Implementation” on page 29](#). This user's guide explains how to install and configure the CICS e*Way.

1.1.1 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 operations and administration; to be thoroughly familiar with CICS and with Windows-style GUI operations.

1.1.2 Components

The CICS e*Way is comprised of the following components:

- The **stcwegenericmonk.exe** file, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk function scripts discussed in [CICS Functions](#) on page 46.

A complete list of installed files appears in [Table 3 on page 7](#).

1.2 System Requirements

The CICS e*Way is available on the following operating systems:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows 2000 (Japanese), Windows 2000 SP1 (Japanese), and Windows 2000 SP2 (Japanese)
- Windows NT 4.0 SP6a
- Windows NT 4.0 SP6a (Japanese)
- Solaris 2.6, 7, and 8
- Solaris 2.6, 7, and 8 (Japanese)
- AIX 4.3.3

To use the CICS e*Way, you need the following:

- An e*Gate Participating Host, version 4.5 or later. For AIX, you need an e*Gate Participating Host, version 4.5.1.
- A TCP/IP network connection.
- 4 MB free disk space for executable, configuration, library, and script files.

The client components of the CICS have their own requirements; see that system's documentation for more details.

1.2.1 External System Requirements

To enable the e*Way to communicate correctly with CICS, you need the following external requirements:

- Supports Universal Client 3.0 and Transaction Gateway 4.0 (Transaction Gateway not supported on Solaris 2.6).
- For CICS running on OS/390, you need a license. Contact your IBM account manager for details.

CICS Universal Client communication with CICS Servers

The tables below use a check mark (✓) to indicate when communication is supported between a CICS Universal Client and a CICS server. See Table 1 “[CICS Universal Client for Solaris Communication with CICS Servers](#)” on page 3 for Solaris communication with CICS servers and Table 2 “[CICS Universal Client for Windows Communication with CICS Servers](#)” on page 3 for Windows communication with CICS servers.

Table 1 CICS Universal Client for Solaris Communication with CICS Servers

| Server | NetBios | TCP/IP | APPC | TCP62 | ECI |
|---|---------|--------|------|-------|-----|
| CICS Transaction Server for OS/2 Version 4.1 | | ✓ | ✓ | | ✓ |
| Transaction Server for Windows NT Version 4 | | ✓ | ✓ | | ✓ |
| TXSeries version 4.2 for Windows NT | | ✓ | ✓ | | ✓ |
| CICS on open systems | | ✓ | ✓ | | ✓ |
| CICS for OS/400 Version 3.1 | | | ✓ | | ✓ |
| CICS for MVS/ESA Version 4.1 | | | ✓ | | ✓ |
| Transaction Server for OS/390 Version 1.x | | | ✓ | | ✓ |
| CICS for VSE/ESA Version 2.3 | | | ✓ | | ✓ |
| CICS Transaction Server for VSE/ESA Version 1.1 | | | ✓ | | ✓ |

Table 2 CICS Universal Client for Windows Communication with CICS Servers

| Server | TCP/IP | APPC | TCP62 | NetBIOS |
|--|-----------------------|------|-------|---------|
| CICS Transaction Server for OS/2 Version 4.1 | ✓ | ✓ | | ✓ |
| Transaction Server for Windows NT Version 4 | ✓ | ✓ | | ✓ |
| TXSeries version 4.2 for Windows NT | ✓ | ✓ | | |
| CICS on open systems | ✓ | ✓ | | |
| CICS for OS/400 Version 3.1 | | ✓ | | |
| CICS for MVS/ESA Version 4.1 | ✓ (See note below) | ✓ | ✓ | |

Table 2 CICS Universal Client for Windows Communication with CICS Servers (Continued)

| Server | TCP/IP | APPC | TCP62 | NetBIOS |
|---|-----------------------|------|-------|---------|
| Transaction Server for OS/390 Version 1.x | ✓ (See note below) | ✓ | ✓ | |
| CICS for VSE/ESA Version 2.3 | | ✓ | | |
| CICS Transaction Server for VSE/ESA Version 1.1 | | ✓ | | |

Note: Use of the TCP/IP protocol for **CICS for MVS/ESA Version 4.1** and **Transaction Server for OS/390 Version 1.x** can only be achieved via the TCP62 protocol. For more information, refer to the IBM documentation for your specific CICS implementation.

Installation

This chapter explains procedures for installing the CICS e*Way.

2.1 Windows NT and Windows 2000

2.1.1 Pre-installation

- Exit all Windows programs before running the setup program, including any anti-virus applications.
- You must have Administrator privileges to install this e*Way.

2.1.2 Installation Procedure

To install the CICS e*Way on a Windows system

- 1 Log in as an Administrator on the workstation on which you want to install the e*Way.
- 2 Insert the e*Way installation CD-ROM into the CD-ROM drive.
- 3 If the CD-ROM drive's Autorun feature is enabled, the setup application launches automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
- 4 The InstallShield setup application launches. Follow the on-screen instructions to install the e*Way.

Be sure to install the e*Way files in the suggested **client** installation directory. The installation utility detects and suggests the appropriate installation directory. **Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.**

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, Intelligent Queues (IQs), and Event Types before this e*Way can perform its intended functions. For more information about any of these procedures, please see the online Help system.

2.2 UNIX

2.2.1 Pre-installation

You do not require root privileges to install this e*Way. Log in under the user name that you wish to own the e*Way files. Be sure that this user has sufficient privileges to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install the CICS e*Way on a UNIX system

- 1 Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
- 2 If necessary, mount the CD-ROM drive.
- 3 At the shell prompt, type
`cd /cdrom`
- 4 Start the installation script by typing
`setup.sh`
- 5 A menu of options will appear. Select the **Install e*Way** option. Then, follow the additional on-screen directions.

Be sure to install the e*Way files in the suggested **client** installation directory. The installation utility detects and suggests the appropriate installation directory. **Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.**

***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 online Help system.*

2.3 Files/Directories Created by the Installation

The CICS e*Way installation process will install the following files, see the table “Files Created by the Installation” on page 7, within the e*Gate directory tree. Files will be installed within the **egate\client** tree on the Participating Host and committed to the **default** schema on the Registry Host.

Table 3 Files Created by the Installation

| e*Gate Directory | File(s) |
|------------------------------------|--|
| \bin\ | stcewgenericmonk.exe stc_monkcics.dll |
| \configs\stcewgenericmonk\ | stcewcics.def |
| \monk_library\monk.Library\ewcics\ | cics-ack.monk cics-connect.monk cics-conn-shutdown.monk cics-exchange.monk cics-init.monk cics-nack.monk cics-notify.monk cics-outgoing.monk cics-shutdown.monk cics-startup.monk cics-verify.monk |

Configuration

You must configure the CICS e*Way before use. This chapter lists all the configuration parameters used by the e*Way together with all supporting information needed, including Monk configuration for connection to the external system.

3.1 e*Way Configuration Parameters

Set the e*Way configuration parameters, using the e*Way Editor.

To change e*Way configuration parameters

- 1 In the Enterprise Manager's Navigator/Component pane, 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** text 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*.

This chapter explains the e*Way's configuration parameters under the following sections:

- General Settings
- Communication Setup
- Monk Configuration
- CICS Settings

3.1.1 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file.

Required Values

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

Additional Information

An Event is journaled for the following conditions:

- When the number of resends is exceeded (see **Max Resends Per Message** below)
- When its receipt is due to an external error, but Forward External Errors is set to **No**. (See [“Forward External Errors” on page 9](#) for more information.)

Max Resends Per Message

Description

Specifies the number of times the e*Way attempts to resend an Event to the external system after receiving an error. When this maximum is reached, the Event 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 Events that the e*Way allows. When the specified number of failed messages is reached, the e*Way shuts down and exits.

Required Values

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

Forward External Errors

Description

Selects whether error messages that begin with the string “DATAERR,” which are received from the external system is queued to the e*Way’s configured queue. See [“Exchange Data with External Function” on page 23](#) for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages are not forwarded. See [“Schedule-driven Data Exchange Functions” on page 17](#) for more information about how the e*Way uses this function.

Communication Setup

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

The schedule you set using the e*Way's properties in the Enterprise Manager controls when the e*Way executable file runs. The schedule you set within the parameters discussed in this section, using the e*Way Editor, determines when data is exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.

Start Exchange Data Schedule

Description

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

Required Values

One of the following values:

- One or more specific dates/times
- A single repeating interval (such as yearly, weekly, monthly, daily, or every n seconds).
- **Also Required** — If you set a schedule using this parameter, you must also define all three of the following functions:
 - ♦ **Exchange Data With External**
 - ♦ **Positive Acknowledgment**
 - ♦ **Negative Acknowledgment**

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

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send a positive or negative acknowledgment to the external system (using the **Positive Acknowledgment** and **Negative Acknowledgment** functions) and whether the connection to the external system is active.

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

See ["Exchange Data with External Function" on page 23](#), ["Exchange Data Interval" on page 11](#), and ["Stop Exchange Data Schedule" on page 10](#) for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following values:

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

Exchange Data Interval

Description

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

Required Values

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

Additional Information

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

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

See [“DownTimeout” on page 11](#) and [“Stop Exchange Data Schedule” on page 10](#) for more information about the data-exchange schedule.

DownTimeout

Description

Specifies the number of seconds that the e*Way waits between calls to the **External Connection Establishment** function. See [“External Connection Establishment Function” on page 24](#) 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 waits between calls to the **External Connection Verification** function. See [“External Connection Verification Function” on page 25](#) 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 waits between attempts to resend a Event (message) to the external system, after receiving an error message from the external system.

Required Values

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

Zero Wait Between Successful Exchanges

Description

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

Required Values

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

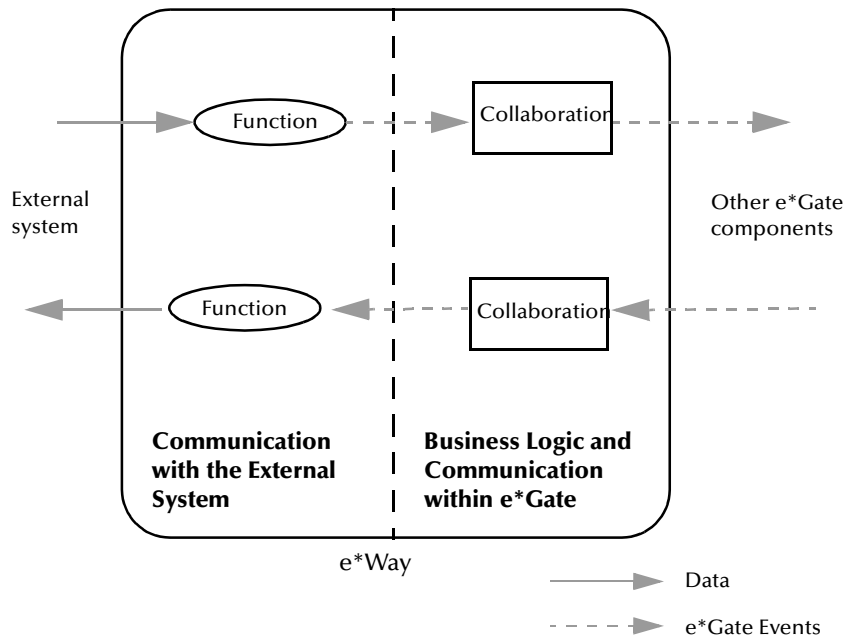
See [“Exchange Data with External Function” on page 23](#) for more information.

3.1.2 Monk Configuration

The parameters in this section help to set up information required by the e*Way to utilize Monk 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 1 on page 13](#)) 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 1 e*Way Internal Architecture



The communications side of the e*Way uses Monk 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 **Monk Configuration** options discussed in this section control the Monk environment and define the Monk functions used to perform these basic e*Way operations. You may create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as **Notepad** or **UNIX vi**).

The communications side 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 Monk environment; therefore, information such as variables, functions, path information, and so on cannot be shared between threads.

Operational Details

The Monk functions in the communications side of the e*Way fall into the groups shown in [Table 4 on page 14](#). A series of figures following the table illustrates the interaction and operation of these functions.

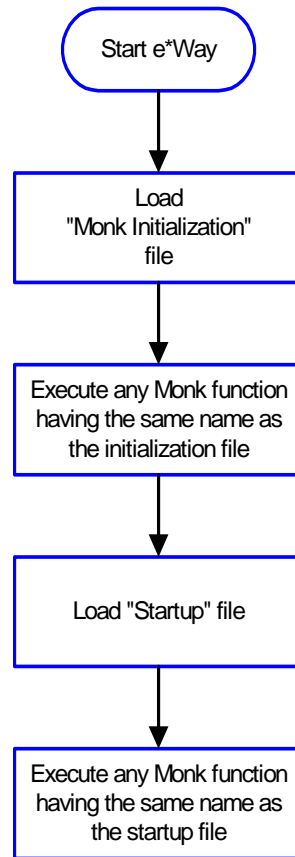
Table 4 Monk Functions, Communications Side

| Type of Operation | Name |
|-------------------------------|---|
| Initialization | Startup Function on page 22 (also see Monk Environment Initialization File on page 22) |
| Connection | External Connection Establishment Function on page 24 External Connection Verification Function on page 25 External Connection Shutdown Function on page 25 |
| Schedule-driven Data Exchange | Exchange Data with External Function on page 23 Positive Acknowledgment Function on page 26 Negative Acknowledgment Function on page 26 |
| Shutdown | Shutdown Command Notification Function on page 27 |
| Event-driven Data Exchange | Process Outgoing Message Function on page 23 |

Initialization Functions

[Figure 2 on page 15](#) illustrates how the e*Way executes its initialization functions.

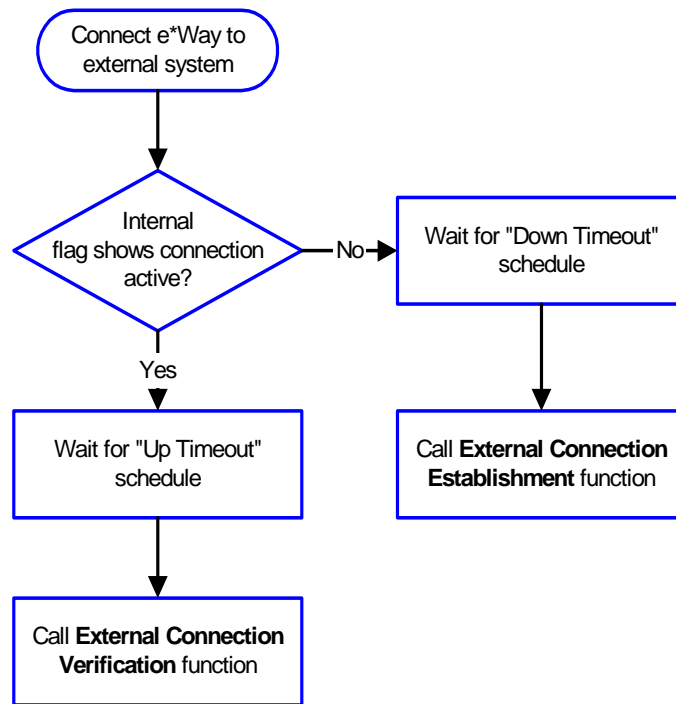
Figure 2 Initialization Functions



Connection Functions

Figure 3 on page 16 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 3 Connection Establishment and Verification Functions

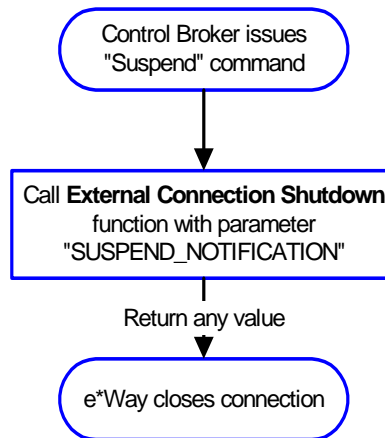


Note: The e*Way selects the connection function based on an internal *up/down* flag rather than a poll to the external system. See [Figure 5 on page 18](#) and [Figure 7 on page 20](#) for examples of how different functions use this flag.

User functions can manually set this flag using Monk functions. See [send-external-up](#) on page 43 and [send-external-down](#) on page 42 for more information.

Figure 4 illustrates how the e*Way executes its **connection shutdown** function.

Figure 4 Connection Shutdown Function



Schedule-driven Data Exchange Functions

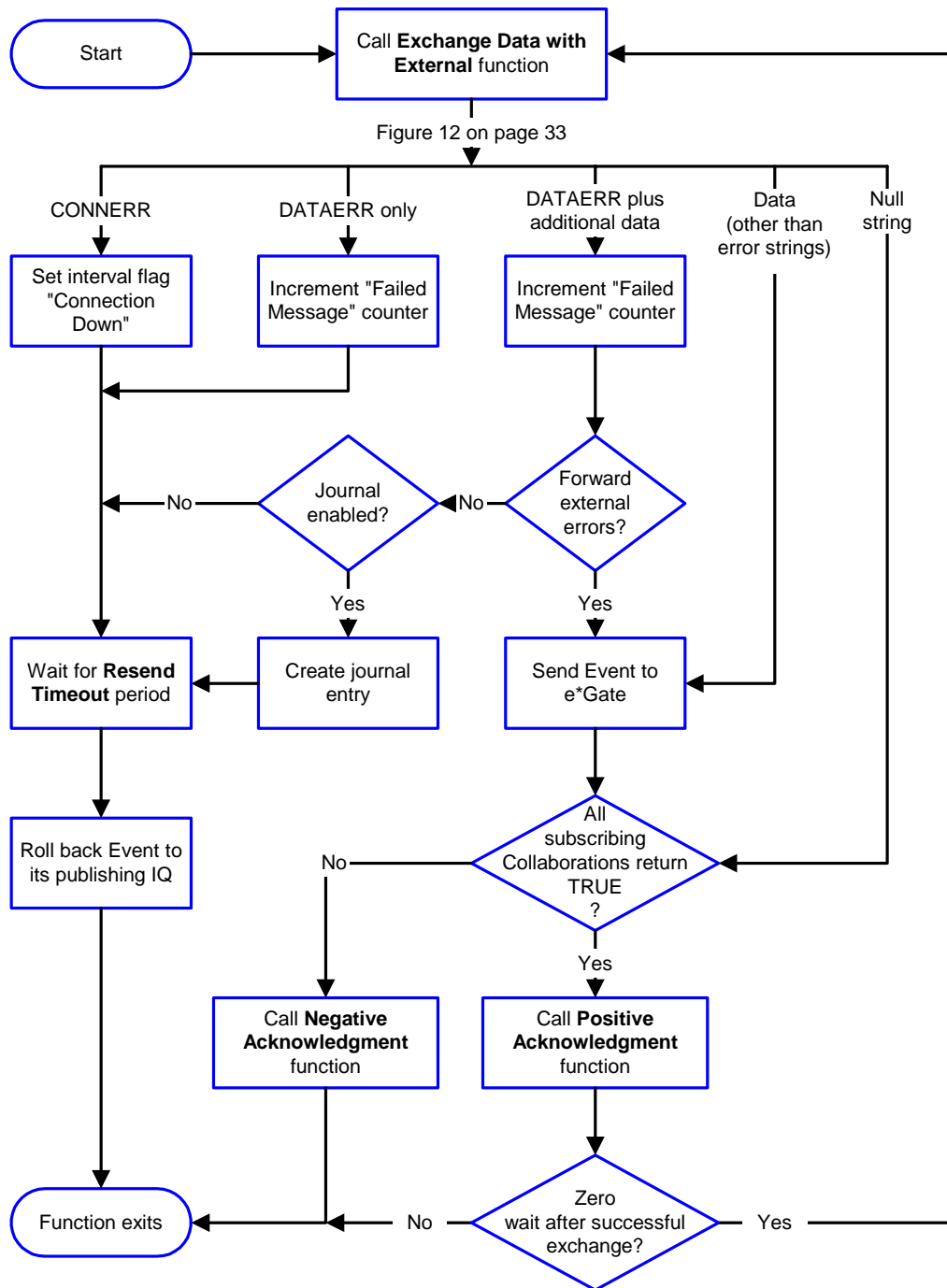
[Figure 5 on page 18](#) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgment Function** and **Negative Acknowledgment Function** are also called during this process.

“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 **Start Data Exchange** time), as set by the **Exchange Data Interval**
- The **start-schedule** Monk function is called

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

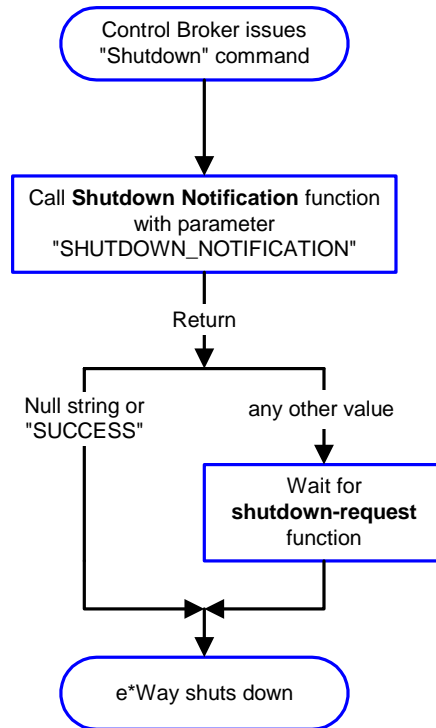
Figure 5 Schedule-driven Data Exchange Functions



Shutdown Functions

Figure 6 below illustrates how the e*Way implements the **shutdown-request** function.

Figure 6 Shutdown Functions



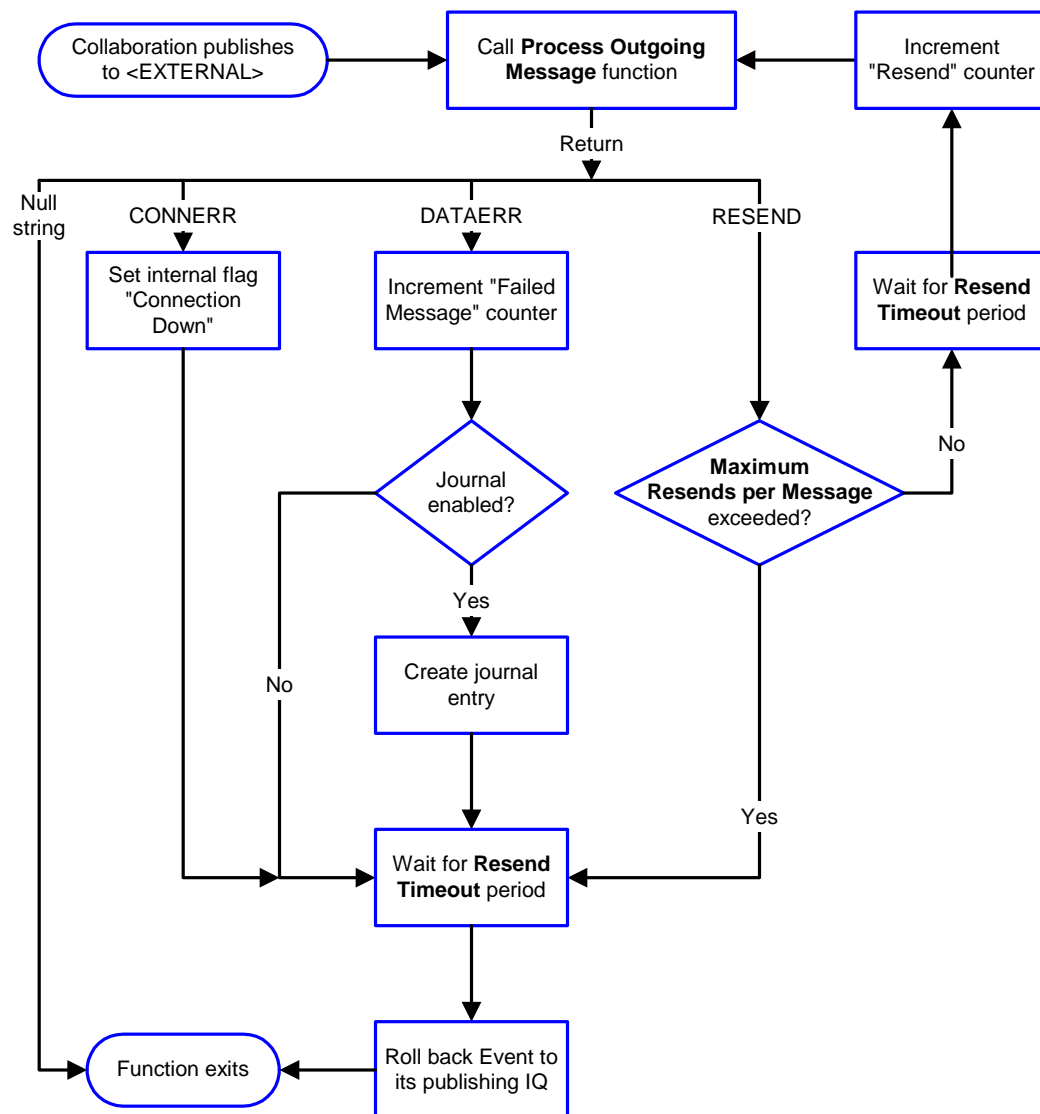
Event-driven Data Exchange Functions

Figure 7 below 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 7 Event-driven Data Exchange Functions



How to Specify Function Names or File Names

Parameters requiring the name of a Monk function accept either a function name or a file name. If you specify a file name, be sure that the file has one of the following extensions:

- **.monk**
- **.tsc**
- **.dsc**

Additional Path

Description

Specifies a path to be appended to the load path, the path Monk uses to locate files and data (set internally within Monk). The directory specified in **Additional Path** is searched after the default load paths.

Required Values

A path name, 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.

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

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

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

Auxiliary Library Directories

Description

Specifies a path to auxiliary library directories. Any **.monk** files found within those directories are automatically loaded into the e*Way's Monk environment. This parameter is optional and may be left blank.

Required Values

A path name, or a series of paths separated by semicolons.

Additional Information

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

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

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

The default is **monk_library/ewcics**.

This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which are loaded after the auxiliary library directories are loaded. Use this feature to initialize the e*Way's Monk environment (for example, to define Monk variables that are used by the e*Way's function scripts).

Required Values

A file name within the load path, or file name plus path information (relative or absolute). If path information is specified, that path is appended to the load path. See [“Additional Path” on page 21](#) for more information about the load path.

Additional Information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way loads this file and tries to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way would attempt to execute the function **my-init**).

Typically, it is a good practice to initialize any global Monk variables that may be used by any other Monk Extension scripts.

The internal function that loads this file is called once when the e*Way first starts up (see [Figure 2 on page 15](#)).

Startup Function

Description

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

Required Values

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

Additional Information

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

The string “FAILURE” indicates that the function failed; any other string (including a null string) indicates success.

This function is called after the e*Way loads the specified Monk Environment Initialization file and any files within the specified auxiliary directories.

The e*Way loads this file and try to invoke a function of the same base name as the file name (see [Figure 2 on page 15](#)). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

Process Outgoing Message Function

Description

Specifies the Monk function responsible for sending outgoing Events (messages) 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 Monk function, or the name of a file (optionally including path information) containing a Monk function. *You may not leave this field blank.*

Additional Information

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination, as specified within the Enterprise Manager (see [Figure 7 on page 20](#) for more details).

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

- Null string — Indicates that the Event was published successfully to the external system.
- “RESEND” — Indicates that the Event should be resent.
- “CONNERR” — Indicates that there is a problem communicating with the external system.
- “DATAERR” — Indicates that there is a problem with the Event (message) data itself.
- If a string other than the following is returned, the e*Way creates 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 Intelligent Queue (IQ), the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See **“event-send-to-egate” on page 40** for more information.*

Exchange Data with External Function

Description

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

Required Values

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

Additional Information

The function accepts no input and must return a string (see [Figure 5 on page 18](#) for more details) as follows:

- Null string — Indicates that the data exchange was completed successfully. No information is sent into the e*Gate system.
- “CONNERR” — Indicates that a problem with the connection to the external system has occurred.
- “DATAERR” — Indicates that a problem with the data itself has occurred. The e*Way handles the string “DATAERR” and “DATAERR” plus additional data differently; see [Figure 5 on page 18](#) for more details.
- 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 Monk function **start-schedule**. After the function has returned **true**, and the data received by this function has been acknowledged by the **Positive Acknowledgment** function or **Negative Acknowledgment** function respectively, the e*Way checks the **Zero Wait Between Successful Exchanges** parameter.

If this parameter is set to **Yes**, the e*Way immediately calls the **Exchange Data with External** function again; otherwise, the e*Way does not call the function until the next scheduled start exchange time or the schedule is manually invoked using the Monk function **start-schedule** (see [start-schedule](#) on page 45 for more information).

External Connection Establishment Function

Description

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

Required Values

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

Additional Information

The function accepts no input and must return a string as follows:

- “SUCCESS” or “UP” — Indicates that the connection was established successfully.
- Any other string (including the null string) — Indicates that the attempt to establish the connection failed.

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 Monk function that the e*Way calls when its internal variables show that the connection to the external system is up.

Required Values

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

Additional Information

The function accepts no input and must return a string as follows:

- “SUCCESS” or “UP” — Indicates that the connection was established successfully.
- Any other string (including the null string) — Indicates that the attempt to establish the connection failed.

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 Monk function that the e*Way calls to shut down the connection to the external system.

Required Values

The name of a Monk function. This parameter is optional.

Additional Information

This function requires a string as input, and may return a string.

This function is only invoked when the e*Way receives a **suspend** command from a Control Broker. When the **suspend** command is received, the e*Way invokes this function, passing the string “SUSPEND_NOTIFICATION” as an argument.

Any return value indicates that the **suspend** command can proceed and that the connection to the external system can be broken immediately.

Positive Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls when *all* the Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk 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 as follows:

- “CONNERR” — Indicates a problem with the connection to the external system. When the connection is re-established, the **Positive Acknowledgment** function is called again, with the same input data.
- Null string — The function completed execution successfully.

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).

Negative Acknowledgment Function

Description

Specifies a Monk function that the e*Way calls when the e*Way fails to process and queue Events from the external system.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk 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 as follows:

- “CONNERR” — Indicates a problem with the connection to the external system. When the connection is re-established, the function is called again.
- Null string — The function completed execution successfully.

This function is only called during the processing of inbound Events. After the **Exchange Data with External** function returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing.

If the Event's processing is not completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the **Negative Acknowledgment** function (otherwise, the e*Way executes the **Positive Acknowledgment** function).

Shutdown Command Notification Function

Description

Specifies a Monk function that is called when the e*Way receives a shutdown command from the Control Broker. This parameter is optional.

Required Values

The name of a Monk function.

Additional Information

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

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

- A null string or "SUCCESS" — Indicates that the shutdown can occur immediately.
- Any other string — Indicates that shutdown must be postponed. Once postponed, shutdown does not proceed until the Monk function **shutdown-request** is executed (see "[shutdown-request](#)" on page 44).

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

3.1.3 CICS Settings

The parameters in this section help you set up the information required by the CICS e*Way.

CICS User

Description

The name of the user on the default CICS system defined in the Universal Client configuration.

CICS User Password

Description

The password of the user on the default CICS system defined in the Universal Client configuration.

3.2 Environment Configuration

To support the operation of this e*Way, no changes are necessary

- In the Participating Host's operating environment
- In the e*Gate system

Note: *Changes to Monk files can be made using the Collaboration Rules Editor (available from within the Enterprise Manager) or with a text editor. However, if you use a text editor to edit Monk files directly, you **must** commit these changed files to the e*Gate Registry or your changes are not implemented.*

*For more information about committing files to the e*Gate Registry, see the Enterprise Manager's online Help system, or the **stcregutil** command-line utility in the e*Gate Integrator System Administration and Operations Guide.*

3.3 External Configuration Requirements

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

Implementation

This chapter covers the modes of operation for the CICS e*Way and instructions for using the Cobol Copybook Converter, a build tool that takes a copybook specification as input and creates an Event Type Definition (ETD).

4.1 Implementation Overview

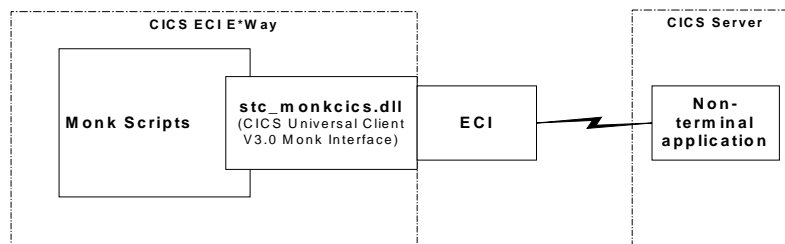
The ECI is an application programming interface (API) to call CICS programs running on a CICS server thereby allowing a non-CICS program to call a CICS program. The calls are the same as if made to a CICS program through the EXEC CICS LINK.

The application does not issue any CICS commands itself; the CICS commands are issued by the called program running in the server. The called program thus appears to have been called by the EXEC CICS LINK with the COMMAREA option, a shared-memory data structure passed between the CICS client application and the CICS transactional program.

The ECI requires that IBM's CICS Universal Client be running on the same computer. The CICS Universal Client version 3 provides the API to the Monk extension **.dll**.

Figure 8 below provides an overview of the architecture.

Figure 8 Architectural Overview



Data Flow

An inbound Event may be sent into the e*Gate environment requesting that a CICS program be called. The input is provided in either a prebuilt Cobol Copybook structure or a proprietary structure. The data is then transformed into an appropriate format and sent to the ECI.

The call returns with the reply from the CICS transaction program. If data is returned, it may be transformed into the necessary structure required by the requester. Finally the transformed data may be presented to the outbound Event.

The CICS e*Way is now ready to process the next request.

4.2 Modes of Operation

The CICS e*Way operates in one of the following modes:

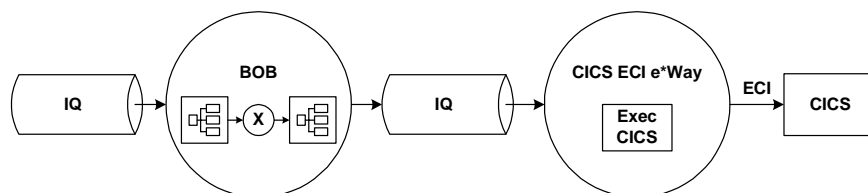
- Outbound to CICS only
- Inbound from CICS only
- Request/reply

This section explains the details of these modes.

Outbound-to-CICS Mode

The outbound e*Way operates in one direction, out of the e*Gate environment and into the CICS environment. No data is expected from the CICS side. See Figure 9 below.

Figure 9 Outbound Mode Example

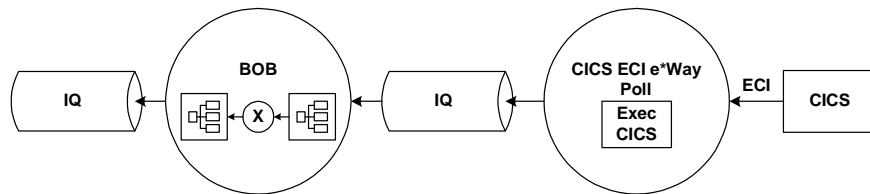


The e*Gate Business Object Broker (BOB) may take an Event from the e*Gate environment and convert it to the Cobol Copybook Event that was generated by Cobol Copybook Converter. The converted Event is then delivered via an Intelligent Queue (IQ) to the e*Way that inserts the copybook Event into the COMMAREA and executes the CICS transaction program. No output from CICS is generated or expected.

Inbound-from-CICS Mode

The inbound only mode uses a time-based polling mechanism to invoke the CICS transaction program. There is no outbound Event coming from the e*Gate environment in this scenario. See Figure 10 below.

Figure 10 Inbound Mode Example

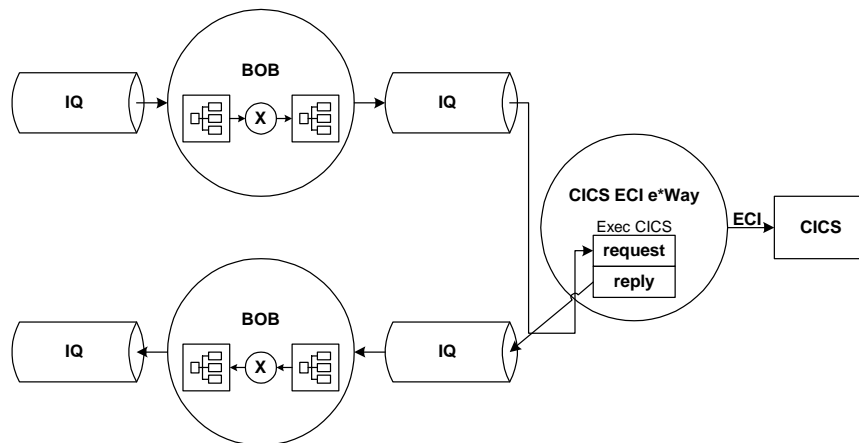


The timer Event triggers from within the e*Way which then signals the invocation of the CICS transaction program. If input was produced, the data is delivered to the inbound BOB via the IQ. The inbound BOB takes the Event and converts it to another format (if necessary) and then forwards the Event to the e*Gate environment.

Request/reply Mode

The request/reply mode is a combination of the inbound and outbound modes. It issues a call to the CICS transaction program then returns the response to the e*Gate environment. See Figure 11 below.

Figure 11 Request/reply Mode Example



This mode operates as follows:

- The outbound BOB takes the Event from the e*Gate environment and may convert it to a layout appropriate to the CICS program being used.
- The Event is sent via IQ to the e*Way that inserts it into the COMMAREA.
- The CICS transaction program is invoked.
- The input is delivered to the inbound BOB via the IQ.
- The inbound BOB takes the Event and converts it (if necessary) before sending it to the e*Gate environment.

4.3 Using the Cobol Copybook Converter

The Cobol Copybook Converter is a build tool that takes a Cobol copybook specification file as input and creates an ETD file, that is, an `.ssc` file. This feature has the following properties:

- You may run this command via the ETD Editor or from the command line.
- The system presents the copybook specification to the Cobol Copybook Converter in a flat file. The converter feature uses the 01 segment of the Cobol copybook as the root node of the ETD.

Note: *For this reason, the input Cobol file must have an 01 segment to operate correctly with the Cobol Copybook Converter.*

For example, after you have generated an ETD file, the e*Gate system can populate the file and present it into the COMMAREA for CICS calls. Similarly, the system can parse the output COMMAREA from CICS into ETDs created by the Cobol Copybook Converter feature.

Cobol Syntax

The input to the Cobol Copybook Converter must conform to standard Cobol syntax rules. Since Cobol's record and field definitions are column-dependent, you must download the input files as text files and preserve their column integrity when editing them with a text editor.

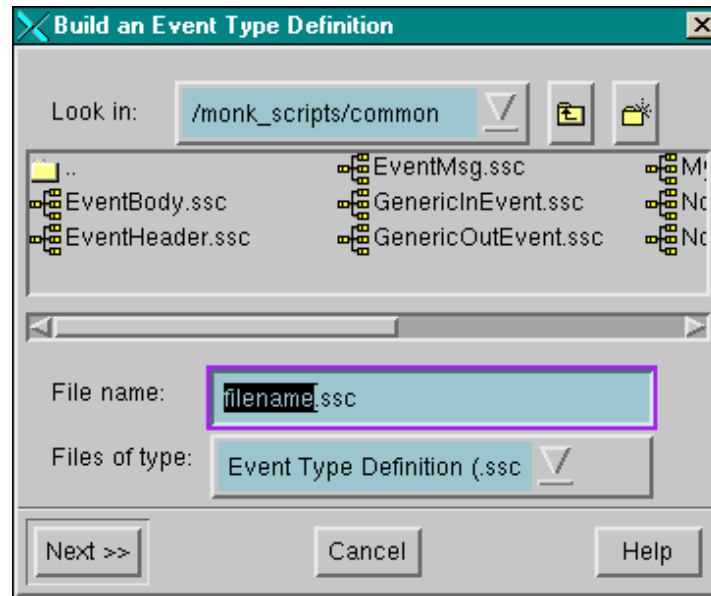
You may access the Cobol Copybook Converter feature via the ETD Editor. Use the procedure below to create an ETD based on the data your installation requires.

To create ETD files using the ETD Editor

- 1 On the ETD Editor window's Toolbar, click **Build**.

The first **Build an Event Type Definition** dialog box appears (see [Figure 12 on page 33](#)).

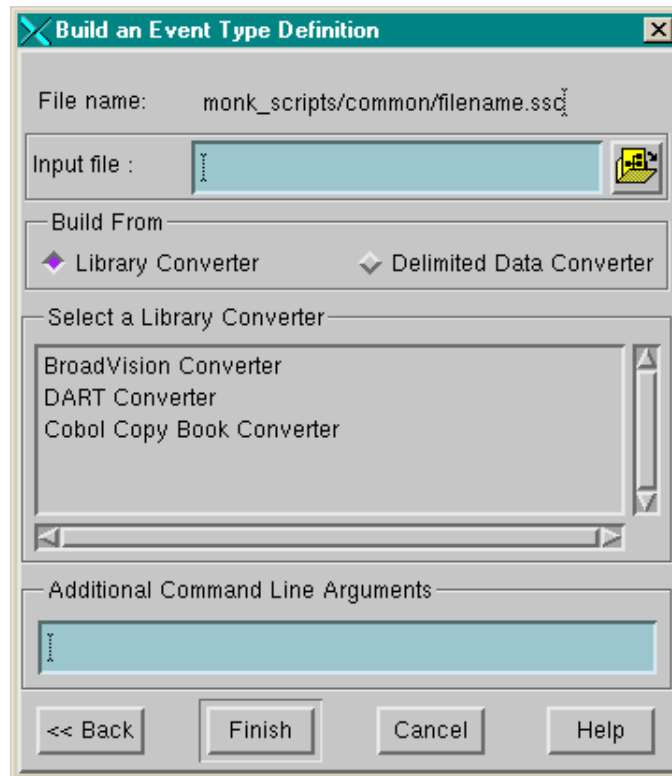
Figure 12 Build an Event Type Definition Dialog Box (Initial)



- 2 In the **File Name** box, type the name of the ETD file you wish to build. *Do not specify any file extension.* The Editor supplies the .ssc extension for you.
- 3 Under **Files of Type**, leave **Event Type Definition** selected.
- 4 Click **Next**.

The second **Build an Event Type Definition** dialog box appears (see [Figure 13 on page 34](#)).

Figure 13 Build an Event Type Definition Dialog Box (Second)



- 5 Type the **Input File** name and path or click the Browse button and navigate to the Input File.
- 6 Under **Build From**, select **Library Converter**.
- 7 Under **Select a Library Converter**, select **Cobol Copy Book Converter**.
- 8 Click **Finish**.

The dialog box closes, and the new ETD appears in the Workspace pane of the ETD Editor window. The conversion process is automatic.

- 9 Click **Save** to save the new ETD.

To create ETD files using the command line

To run the Cobol Copybook Converter from the command line, as a batch utility, enter:

```
stccococo [-I] [-d] -i <copybook_file> <etd>.ssc
```

The following table explains these command options:

| Command | Type | Description |
|---------------|--------|---|
| -I | string | An optional parameter that indicates ICL data origin. |
| -d | string | An optional parameter that indicates DBCS input data mode. |
| -nls | string | Allows the conversion of Cobol files written in Japanese; currently the only string supported is SJIS (Japanese character set). Use of any other (or no) parameter defaults to English/ASCII. |
| -i | string | A mandatory parameter. |
| copybook_file | path | The relative path of the file to be used as the input (mandatory). |
| etd | string | The name of the ETD file you wish to create (mandatory). |

Note: If you enter *stccococo -h* at the command line, seeking help, you get the following message:

```
USAGE: stccococo [-I] [-d] [-c control [-s section]] [-nls <character set>] -i  
copybook_file <etd> .ssc
```

-I indicates ICL data origin. *-d* indicates DBCS mode. Ignore references to the *-c* and *-s* flags. They are for future use only and do not operate in the current system.

For more information on how to use this feature, see the *Cobol Copybook Converter User's Guide*.

4.4 ECI Error Codes

Table 5 on page 36 contains a list of the possible error codes returned by the ECI. These codes are defined in the file **cics-init.monk** which is loaded when the CICS e*Way is initialized. For more information, see **"Monk Environment Initialization File" on page 22**.

For more information on the CICS ECI error codes, see *CICS Family: Client/Server Programming* by IBM Corp. (document number SC33-1435-03). This can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DFHZAD13/.

Note: The file **cics-init.monk** also initializes other values in addition to the error codes listed below. For a complete list of the values initialized by this function, use a text editor to view the **cics-init.monk** file.

Table 5 ECI Error Codes

| ECI Return Value | Error | Description |
|------------------|-----------------------------|--|
| 0 | ECI_NO_ERROR | The call completed successfully. |
| -1 | ECI_ERR_INVALID_DATA_LENGTH | The value in eci_commarea_length field is outside the valid range, or is inconsistent with the value in eci_commarea , being zero for a non-null eci_commarea pointer, or non-zero for a null eci_commarea pointer. |
| -2 | ECI_ERR_INVALID_EXTEND_MODE | The value in eci_extend_mode field is not valid. |
| -3 | ECI_ERR_NO_CICS | The client is unavailable, the server implementation is unavailable, or a logical unit of work was to be started (but the CICS server specified in eci_system_name is not available). No resources have been updated. |
| -4 | ECI_ERR_CICS_DIED | A logical unit of work was to be started or continued, but the CICS server was no longer available. If eci_extend_mode was ECI_EXTENDED, the changes are rolled back, and the logical unit of work ends. If eci_extend_mode was ECI_NO_EXTEND, ECI_COMMIT, or ECI_BACKOUT, the application cannot determine whether the changes have been committed or rolled back, and must log this condition to aid future manual recovery. |
| -5 | ECI_ERR_REQUEST_TIMEOUT | The time-out interval expired before the request could be processed, or the specified interval was negative. |
| -5 | ECI_ERR_NO_REPLY | There was no outstanding reply. |
| -6 | ECI_ERR_RESPONSE_TIMEOUT | The time-out interval expired while the program was running. |
| -7 | ECI_ERR_TRANSACTION_ABEND | The CICS transaction that executed the requested program abended (ended abnormally). The abend code will be found in eci_abend_code . For information about abend codes and their meaning, consult the documentation for the server system to which the request was directed. |

Table 5 ECI Error Codes

| ECI Return Value | Error | Description |
|------------------|---|---|
| -8 | ECI_ERR_EXEC_NOT_RESIDENT and ECI_ERR_LUW_TOKEN | The value supplied in eci_luw_token is invalid. |
| -9 | ECI_ERR_SYSTEM_ERROR | An internal system error occurred. The error might have been in the client or in the server. The programmer should save the information returned in the eci_sys_return_code field, as this will help service personnel to diagnose the error |
| -10 | ECI_ERR_NULL_WIN_HANDLE | An asynchronous call was specified with the window handle set to 0. |
| -12 | ECI_ERR_NULL_MESSAGE_ID | An asynchronous call was specified with the message identifier set to 0. |
| -13 | ECI_ERR_THREAD_CREATE_ERROR | The server implementation or the client failed to create a thread to process the request. |
| -14 | ECI_ERR_INVALID_CALL_TYPE | The call type was not one of the valid call types. |
| -15 | ECI_ERR_ALREADY_ACTIVE | An attempt was made to continue an existing logical unit of work, but there was an outstanding asynchronous call for the same logical unit of work. |
| -16 | ECI_ERR_RESOURCE_SHORTAGE | The server implementation or the client did not have enough resources to complete the request. |
| -17 | ECI_ERR_NO_SESSIONS | A new logical unit of work was being created, but the application already has as many outstanding logical units of work as the configuration will support. |
| -18 | ECI_ERR_NULL_SEM_HANDLE | A null semaphore handle was passed when a valid handle was required. |
| -19 | ECI_ERR_INVALID_DATA_AREA | Either the pointer to the ECI parameter block is invalid, or the pointer supplied in eci_commarea is invalid. |
| -21 | ECI_ERR_INVALID_VERSION | The value supplied for eci_version was invalid. |
| -22 | ECI_ERR_UNKNOWN_SERVER | The requested server could not be located. Only servers returned by CICS_EciListSystems are acceptable. |
| -23 | ECI_ERR_CALL_FROM_CALLBACK | The call was made from a callback routine. |

Table 5 ECI Error Codes

| ECI Return Value | Error | Description |
|------------------|-------------------------|---|
| -24 | ECI_ERR_INVALID_TRANSID | A logical unit of work was being extended, but the value supplied in eci_transid differed from the value used when the logical unit of work was started. |
| -25 | ECI_ERR_MORE_SYSTEMS | There was not enough space in the List array to store the information. The supplied array has been filled and the systems parameter has been updated to contain the total number of systems found so that an array of suitable size can be allocated and the function can be retried. |
| -26 | ECI_ERR_NO_SYSTEMS | No CICS servers can be located. In this case, the value returned in Systems is zero. |
| -27 | ECI_ERR_SECURITY_ERROR | An invalid combination of user ID and password was supplied. |
| -28 | ECI_ERR_MAX_SYSTEMS | Requests were made to more servers than the configuration allows. Consult the documentation for your client or server for information on controlling the number of servers that can be used. |
| -29 | ECI_ERR_MAX_SESSIONS | There were not enough communication resources to satisfy the request. Consult the documentation for your client or server for information on controlling communication resources. |
| -30 | ECI_ERR_ROLLEDBACK | An attempt was made to commit a logical unit of work, but the server was unable to commit the changes and backed them out instead. |

CICS e*Way Functions

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

- **Basic Functions** on page 39
- **CICS Functions** on page 46

The functions explained in this chapter control the CICS e*Way's basic operations, as well as those needed to interface with the CICS application.

***Note:** These functions can only be used by the functions defined within the e*Way's configuration file. None of the functions are available to Collaboration Rules scripts executed by the e*Way.*

5.1 Basic Functions

The functions in this category control the e*Way's most basic operations.

The basic functions are:

- event-send-to-egate** on page 40
- get-logical-name** on page 41
- send-external-down** on page 42
- send-external-up** on page 43
- shutdown-request** on page 44
- start-schedule** on page 45
- stop-schedule** on page 46

event-send-to-egate

Syntax

(event-send-to-egate *string*)

Description

event-send-to-egate sends data that the e*Way has already received from the external system into the e*Gate system as an Event.

Parameters

| Name | Type | Description |
|--------|--------|---|
| string | String | The data to be sent to the e*Gate system. |

Return Values

Boolean

Returns true (**#t**) if the data is sent successfully; otherwise, returns false (**#f**).

Throws

None.

Additional information

This function can be called by any e*Way function when it is necessary to send data to the e*Gate system in a blocking fashion.

get-logical-name

Syntax

(get-logical-name)

Description

get-logical-name returns the logical name of the e*Way.

Parameters

None.

Return Values

string

Returns the name of the e*Way (as defined by the Enterprise Manager).

Throws

None.

send-external-down

Syntax

(send-external-down)

Description

send-external down instructs the e*Way that the connection to the external system is down.

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

(send-external-up)

Description

send-external-up instructs the e*Way that the connection to the external system is up.

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

(shutdown-request)

Description

shutdown request requests the e*Way to perform the shutdown procedure when there is no outstanding incoming/outgoing event. When the e*Way is ready to act on the shutdown request, it invokes the Shutdown Command Notification Function (see [“Shutdown Command Notification Function” on page 27](#)). Once this function is called, the shutdown proceeds immediately.

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

Parameters

None.

Return Values

None.

Throws

None.

start-schedule

Syntax

(start-schedule)

Description

start-schedule requests that the e*Way execute the **Exchange Events with External** function specified within the e*Way's configuration file. Does not affect any defined schedules.

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

(stop-schedule)

Description

stop-schedule requests that the e*Way halt execution of the **Exchange Events with External** function specified within the e*Way's configuration file. Execution is stopped when the e*Way concludes any open transaction. Does not affect any defined schedules, and does not halt the e*Way process itself.

Parameters

None.

Return Values

None.

Throws

None.

5.2 CICS Functions

The two functions in the .dll file that can be invoked from Monk are:

[EciListSystems](#) on page 47

[ExternalCall](#) on page 48

EciListSystems

Syntax

```
(invoke obj "EciListSystems")
```

Description

EciListSystems determines which CICS servers are defined in the Universal Client configuration; *obj* is the Monk object returned from load-interface. See the example below.

Parameters

None.

Return Values

Vector of Monk strings

An IBM Universal client installation has a notion of a default CICS server, and this is guaranteed to be the first item in the return vector.

Throws

If no servers can be found, an exception is generated.

Example

The following printout shows an example of how to list all CICS servers in the Universal Client configuration:

```
(let ((cics (load-interface "stc_monkcics.dll" "init_cicsext"))) ;  
Load DLL and initialize  
  (let ((sv (invoke cics "EciListSystems"))) ; Invoke  
EciListSystems  
    (do ((i 0 (+ i 1))) ; Begin loop  
        ((>= i (vector-length sv))) ; Loop control  
          (display (vector-ref sv i)) ; Display the specified            (newline) ; Write a newline character  
          )  
    )  
  )
```

Note: *The existence of a CICS server does not guarantee that it can be contacted.*

ExternalCall

Syntax

```
(invoke obj "ExternalCall" eci_call_type eci_program_name eci_userid
eci_password eci_transid comm eci_extend_mode eci_luw_token
eci_version eci_system_name eci_tpn N)
```

Description

ExternalCall is the one and only interface to CICS using the External Call Interface (ECI) API.

Parameters

| Parameter | Description | Required Values |
|------------------------------|--|---|
| eci_call_type | Specifies whether the ECI call type is Asynchronous or Synchronous. <ul style="list-style-type: none"> ▪ Synchronous Calls will wait for the transaction to complete, then return the contents of the COMMAREA. ▪ Asynchronous calls will not wait for the transaction to complete, so no data is returned. For further detail, see the IBM publication "CICS Family: Client/Server Programming" (document number SC33-1435-03). | Asynchronous or Synchronous. Synchronous is the configured default. |
| eci_program_name | Specifies the CICS program to be run on the server. Maximum length is eight characters. | A valid CICS program name, eight characters or less. |
| eci_userid | Specifies the ID of the CICS user. Maximum length is eight characters. | A valid CICS user ID, eight characters or less. |
| eci_password | Specifies the password for the CICS user. Maximum length is eight characters. | A valid password for the user ID, eight characters or less. |
| eci_transid | Specifies the CICS TransId to be run on the server. Maximum length is four characters. | A valid CICS TransId, four characters or less. |
| comm - comm area (send) data | A Monk object whose payload data (potentially binary) is to be transmitted to the specified CICS program. | |

Parameters

| Parameter | Description | Required Values |
|----------------------|---|--|
| eci_extend_mode | <p>Specifies the extended mode to be requested of the CICS server.</p> <ul style="list-style-type: none"> No (ECI_NO_EXTEND). IF the input eci_luw_token field is zero, this will be the only call for a logical unit of work. If the input eci_luw_token field is not zero, this will be the last call for the specified logical unit of work. In either case, changes to recoverable resources are committed by a CICS end-of-task syncpoint, and the logical unit of work ends. <p>Yes (ECI_EXTENDED). If the input eci_luw_token field is zero, this will be the first call for a logical unit of work that is to be continued. If the input eci_luw_token field is not zero, this call will continue the specified logical unit of work. In either case the logical unit of work continues after the called program completes, and changes to recoverable resources remain uncommitted.</p> | Yes or No. The configured default is No. |
| eci_luw_token | <p>Specifies the logical unit of work to which a call belongs. This must be set to zero at the start of a logical work unit. The ECI will update the value on the first or only call of the logical work unit. If the unit of work is to be extended, this value should be used as input to all subsequent calls associated with the same logical work unit. If the return code is not ECI_NO_ERROR and a call is ending or continuing an existing logical work unit, then this field will be used to report the state of the logical work unit. If it is zero, the logical work unit has ended and updates have been backed out. If it is not zero, the value is the same as the input value. The logical work unit is continuing, and updates are still pending.</p> | A valid integer in the range of 0 to 1000. The configured default is 0. This is a required input and output parameter. |
| eci_version | Version of ECI running on the CICS system in eci-system_name. | |
| eci_system_name | The CICS system on which the program that you want to send data to is running. | |
| eci_tpn | Reserved for future use. This parameter must be set to a null value ("") at this time. | |
| N - Comm area length | Specifies the length (in bytes) of the communication area (COMMAREA) passed to the ECI. | An integer in the range of 1 to 32659. The configured default is 1000. |

Note: Programming and CICS Universal Client Online Library. These documents are available online from the following URLs:

<http://www.s390.ibm.com/bookmgr-cgi/bookmgr.exe/BOOKS/DFHZAD13/CONTENTS>

For information on CICS Transaction Gateway Version 4.0 visit:

<http://www-4.ibm.com/software/ts/cics/library/manuals>

This function has the following properties:

- The **Comm area (send) data** and the **Comm area (receive) length** have to be specified separately because the behavior of the called program can only be known by the caller. SeeBeyond's implementation cannot determine whether a specified program requires data to be sent to it, or if it returns data (as in the example given at the end of this section) or possibly both.
- Internally, the **Comm area** is set to a size which is the larger of the value given in **Commarea length** and the size of the object in **Commarea (send) data**. It is therefore guaranteed to be large enough to accommodate both the data to be sent and any data returned.

Return Values

Vector of Monk strings

The vector returned by **ExternalCall** comprises the following elements:

- The status code returned by the C API function **CICS_ExternalCall**.

If this value is equal to **ECI_NO_ERROR** the call has been successful. For detailed analysis, the Monk programmer should consider making comparisons against the other defined values with names beginning **ECI_ERR_**. For more information on ECI return codes, see "[ECI Error Codes](#)" on page 35.

- An abend code of four characters (space padded).
- The contents of the **comm area**.

This may not be relevant; it depends on the behavior of the CICS program but it is always returned.

- The contents of **eci_sys_return_code**.

This element only needs to be considered if the status code in element 1 is equal to **ECI_ERR_SYSTEM_ERROR**.

- The contents of **eci_luw_token**.

If a transaction is started by the most recent invocation of **ExternalCall**, this value is needed for subsequent calls within the same transaction.

Throws

An exception is thrown if:

- The number of parameters is incorrect
- Any parameters are of the wrong type or length

- If the size of the object in **Comm area (send) data** or the size specified in **Comm area length** is larger than the maximum size of the comm area, that is, 32,500 characters.
- The maximum length of a user ID or password is exceeded. User names and passwords are limited to eight characters in **ECI_VERSION_0**; otherwise they are limited to 16 characters.

See the sample printout under **Example** on page 51.

Example

```
(define ECI_VERSION_1A 2)
(define ECI_NO_EXTEND 0)
(define ECI_SYNC 516)

(let ((cics (load-interface "stc_monkcics.dll" "init_cicsext")))
  (let ((sv (cics "EciListSystems")))
    (display (cics "ExternalCall"
                  ECI_SYNC           ; eci_call_type
                  "ECPROG"          ; eci_program_name
                  "sysad"           ; eci_userid
                  "sysad"           ; eci_password
                  "EC01"            ; eci_transid
                  " "                ; Comm area (send) data
                  ECI_NO_EXTEND     ; eci_extend_mode
                  0                  ; eci_luw_token
                  ECI_VERSION_1A    ; eci_version
                  (vector-ref sv 0) ; eci_system_name
                  " "                ; eci_tpn
                  18))              ; Comm area (receive) length
      (newline)
    )
  )
)
```

In practice, a start-up Monk file is set up with the three defined values at the beginning of this example for all possible values needed to run in ECI mode.

Index

A

- Additional Path
 - configuration parameter 21
- Architectural Overview 29
- Auxiliary Library Directories
 - configuration parameter 21

B

- basic functions
 - event-send-to-egate 40
 - get-logical-name 41
 - listed 39
 - send-external-up 43
 - shutdown-request 44
 - chart of 19
 - start-schedule 45
 - stop-schedule 46

C

- CICS
 - described 1
- CICS e*Way
 - changing configuration parameters 8
 - components 2
 - defined 1
 - external system requirements 2
 - functions
 - basic functions listed 39
 - categories of 39
 - CICS functions listed 46
 - connection shutdown 17
 - modes of operation 30
 - inbound-from-CICS mode 31
 - outbound-to-CICS mode 30
 - request/reply mode 31
 - parameter settings 27
 - UNIX installation 6
 - Windows installation 5
- CICS functions
 - EciListSystems 47
 - ExternalCall 48
- CICS servers

- communication with
 - CICS Universal Client for Solaris 3
 - CICS Universal Client for Windows 3
- CICS User ID
 - configuration parameter 27
- CICS User Password
 - configuration parameter 27
- Cobol 29
- Cobol Copybook Converter
 - conform to Cobol syntax 32
 - function described 1
 - using 32
- Cobol syntax 32
- command line
 - creating ETD files 35
 - description of options 35
- COMMAREA 1, 29
- components
 - of CICS e*Way 2
- configuration parameters 8–27
 - Additional Path 21
 - Auxiliary Library Directories 21
 - changing 8
 - CICS User ID 27
 - CICS User Password 27
 - Down Timeout 11
 - Exchange Data Interval 11
 - Exchange Data with External Function 23
 - External Connection Establishment Function 24
 - External Connection Shutdown Function 25
 - External Connection Verification Function 25
 - Forward External Errors 9
 - Journal File Name 9
 - Max Failed Messages 9
 - Max Resends Per Message 9
 - Monk Environment Initialization File 22
 - Negative Acknowledgment Function 26
 - Positive Acknowledgment Function 26
 - Process Outgoing Message Function 23
 - Resend Timeout 12
 - Shutdown Command Notification Function 27
 - Start Exchange Data Schedule 10
 - Startup Function 22
 - Stop Exchange Data Schedule 10
 - Up Timeout 11
 - Zero Wait Between Successful Exchanges 12
- Connection functions
 - Establishment and Verification
 - chart of 16
 - listed
 - External Connection Establishment 14
 - External Connection Shutdown 14
 - External Connection Verification 14
 - Shutdown function

Index

- chart of 17
- connection shutdown
 - CICS e*Way function 17
- Customer Information Control System
 - described 1

D

- Data Exchange functions
 - Schedule-driven 17
 - chart of 17, 18
- directories
 - created by installation 7
- Down Timeout
 - configuration parameter 11

E

- ECI
 - defined 29
- EciListSystems
 - CICS function 47
- ETD Editor
 - creating ETD files 32
- ETD files
 - creating using the command line 35
 - creating with ETD Editor 32
- Event-driven Data Exchange
 - Process Outgoing Message 14
- Event-driven Data Exchange functions
 - chart of 20
- event-send-to-egate
 - basic function 40
- Exchange Data Interval
 - configuration parameter 11
- Exchange Data with External Function
 - configuration parameter 23
- External Connection Establishment Function
 - configuration parameter 24
- External Connection Shutdown Function
 - configuration parameter 25
- External Connection Verification Function
 - configuration parameter 25
- external system requirements
 - CICS e*Way 2
- ExternalCall
 - CICS function 48

F

- file names
 - how to specify 21
- files

- created by installation 7
- Forward External Errors
 - configuration parameter 9
- function names
 - how to specify 21

G

- get-logical-name
 - basic function 41

I

- Initialization functions
 - chart of 15
 - listed
 - Startup 14
- installation
 - directories created by 7
 - files created by 7

J

- Journal File Name
 - configuration parameter 9

M

- Max Failed Messages
 - configuration parameter 9
- Max Resends Per Message
 - configuration parameter 9
- modes of operation
 - CICS e*Way 30
 - inbound-from-CICS mode 31
 - outbound-to-CICS mode 30
 - request/reply mode 31
- Monk Environment Initialization File
 - configuration parameter 22
- Monk functions
 - communications side
 - types of operations listed 14

N

- Negative Acknowledgment Function
 - configuration parameter 26

P

- parameters - *See* configuration parameters.
- Positive Acknowledgment Function
 - configuration parameter 26

Index

pre-installation
 UNIX 6
 Windows NT 5
Process Outgoing Message Function
 configuration parameter 23

R

Resend Timeout
 configuration parameter 12

S

Schedule-driven Data Exchange functions 17
 chart of 17, 18
 listed
 Exchange Data with External 14
 Negative Acknowledgment 14
 Positive Acknowledgment 14
schedules 45, 46
send-external-up
 basic function 43
Shutdown Command Notification Function
 configuration parameter 27
Shutdown functions
 chart of 19
 listed
 Shutdown Command Notification 14
shutdown-request
 basic function 44
 chart of 19
Start Exchange Data Schedule
 configuration parameter 10
start-schedule
 basic function 45
Startup Function
 configuration parameter 22
Stop Exchange Data Schedule
 configuration parameter 10
stop-schedule
 basic function 46

U

UNIX
 CICS e*Way installation 6
 pre-installation 6
Up Timeout
 configuration parameter 11

W

Windows NT

CICS e*Way installation 5
Windows NT 4.0
 pre-installation 5

Z

Zero Wait Between Successful Exchanges
 configuration parameter 12