SeeBeyond ICAN Suite

e*Way Intelligent Adapter for Jacada Enterprise/Access User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Monk Version

Version 20050406093423.

Contents

Chapter 1

Introduction

The e*Way Intelligent Adapter for Jacada Enterprise/Access provides a means to implement automatic data interchange with an Enterprise Access application running on an external system for which APIs are not readily available. This document describes how to install and configure the Jacada Enterprise/Access e*Way.

1.1 Overview

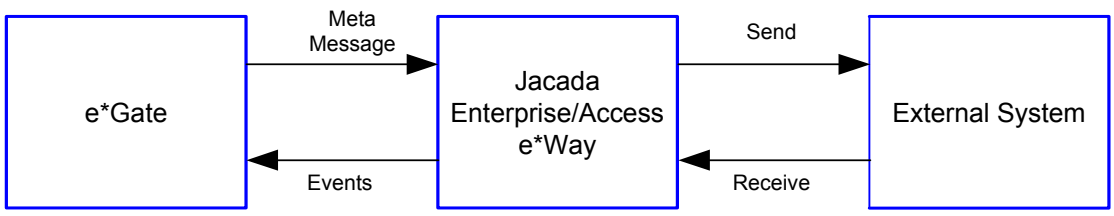
The Jacada Enterprise/Access e*Way provides data translation and manipulation of a binary stream intended to interact with a person, in the form of a monitor screen and keyboard, and matches the data content to the screen order.

Entering information into the application requires transmission of a serial data stream that emulates keyboard entry in response to screen prompts. This transmission requires specific knowledge of the nature of each data element as it relates to the screen position. This process is referred to as *screen scraping*.

The Jacada Enterprise/Access Integrator application appears to read data from a user's monitor during the "screen scraping" process. In a typical application, (for example, a medical-records database), data is displayed on the user's monitor, while the user interacts with the application using a keyboard. Each data element (for example, a patient name, an ID number, or a service charge code) will be displayed on the user's monitor in a known position. The "screen scraper" intercepts the data stream that sends information to the monitor. It then uses the monitor-position information to locate and capture the data being sent to the monitor from the data stream. Although the data is not literally read from or displayed onto the screen, the result is the same.

Data flow between e*Gate, the Jacada Enterprise/Access e*Way, and the external system is depicted in the following diagram:

Figure 1 Data Flow



e*Gate initiates the session with the external system after creating a Meta Data Structure (Event) and sending it to the e*Way as a trigger. This Event contains all the information necessary for the e*Way to communicate with the external system. This information can be used to initiate, respond to, or validate the data interchange.

The Jacada Enterprise/Access e*Way provides an **invoke** function that takes in methods and parameters which in turn enable interaction with the E/A service object.

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 be experienced in Windows operations and administration; to be thoroughly familiar with *Jacada Enterprise/Access Integrator applications*; and to have a working understanding of Windows-style GUI operations.

1.1.2. Components

The Jacada Enterprise/Access e*Way comprises the following:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk function scripts, discussed in **"Jacada Enterprise/Access e*Way Functions" on page 48**.

A complete list of installed files appears in **Table 1 on page 12**.

1.2 Supported Operating Systems

The Jacada Enterprise/Access e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP-UX 11.0 and 11i (PA-RISC)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9

1.3 System Requirements

To use the Jacada Enterprise/Access e*Way, you need the following:

- An e*Gate Participating Host.
- 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.

The client components of Jacada Enterprise/Access Integrator have their own requirements; see that system's documentation for more details.

1.3.1. External System Requirements

The Java-enabled Jacada Enterprise/Access e*Way requires the following installed on the participating host:

- Jacada Enterprise/Access Integrator Version 3.5 and 4.0

To enable the e*Way to communicate properly with the Jacada Enterprise/Access Application Version 3.5, the following are required:

For Windows 2000

- Jacada Enterprise/Access Integrator "client"(specifically the client run-time dll, **EACIw32.dll**, must be present.
- Append **C:\EA2000_3018\lib**, *(verify the correct path first)* to the actual path in the system environment variables.

For Unix

- Append .../lib *(verify the correct path first)* to the actual path in the system environment variables.

Note: *Jacada Integrator 4.0 is only supported for features which are compatible with version 3.5. New features for version 4.0 are not currently supported.*

Chapter 2

Installation

This chapter describes how to install the Jacada Enterprise/Access e*Way.

2.1 Windows Installation

2.1.1. Pre-installation

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

2.1.2. Installation Procedure

To install the Jacada Enterprise/Access 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 should launch 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 will launch. Follow the on-screen instructions to install the e*Way.

Note: *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.***

5. After the installation is complete, exit the install utility and launch the Schema Designer.
6. In the Component editor, create a new e*Way.
7. Display the new e*Way's properties.

8 On the General tab, under **Executable File**, click **Find**.

9 Select the file **stcewgenericmonk.exe**.

10 Under **Configuration file**, click **New**.

11 From the **Select an e*Way template** list, select **stcewcntea** and click **OK**.

12 The e*Way Editor will launch. Make any necessary changes, then save the configuration file.

13 You will return to the e*Way's property sheet. Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 3**.

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.*

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

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 privilege to create files in the e*Gate directory tree.

2.2.2. Installation Procedure

To install the Jacada Enterprise/Access 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/setup

4 Start the installation script by typing:

setup.sh

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

Note: *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.

6 After installation is complete, exit the installation utility and launch the Schema Designer.

7 In the Component editor, create a new e*Way.

8 Display the new e*Way's properties.

9 On the **General** tab, under **Executable File**, click **Find**.

10 Select the file **stcewgenericmonk.**

11 Click **OK** to close the properties sheet, or continue to configure the e*Way. Configuration parameters are discussed in **Chapter 3**.

*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.*

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

2.3 Files/Directories Created by the Installation

The Jacada Enterprise/Access e*Way installation process will install the following files 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 1 Files created by the installation

e*Gate Directory	File(s)
bin\	stcewgenericmonk.exe stccnteaconvert.exe
bin\	stc_monkcntea.dll
configs\stcewgenericmonk\	stcewcntea.def
monk_library\ewcntea\	cntea-ack.monk cntea-call-struct.monk cntea-exchange.monk cntea-extconnect.monk cntea-init.monk cntea-nack.monk cntea-notify.monk cntea-outgoing.monk cntea-shutdown.monk cntea-startup.monk cntea-verify.monk stccicsamnu-pomf.tsc cicsamnu_add.tsc cicsamnu_browse.tsc cicsamnu_inquire.tsc cicsamnu_update.tsc
monk_scripts\common\	cntearequest.ssc gcntearesult.ssc stc_cicsamnu2.ssc

Chapter 3

Configuration

This chapter describes how to configure the Jacada Enterprise/Access e*Way.

3.1 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 *Working with e*Ways* user's guide.

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

- General Settings
- Communication Setup
- Monk Configuration
- CNTEA Configuration

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 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 14** 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.

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 "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 28** 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 22** for information about how the e*Way uses this function.

3.1.2. 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 16** and **"Stop Exchange Data Schedule" on page 16** 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 28** 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 all three of the following:

- Exchange Data With External Function
- Positive Acknowledgment Function
- Negative Acknowledgment Function

If you do not do so, the e*Way 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 ACK or NAK to the external system (using the Positive and Negative Acknowledgment functions) and whether the connection to the external system is active. If no ACK/NAK is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External** function. Thereafter, the **Exchange Data with External** function 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 28**, **"Exchange Data Interval" on page 15**, and **"Stop Exchange Data Schedule" on page 16** 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 29** 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 29** 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

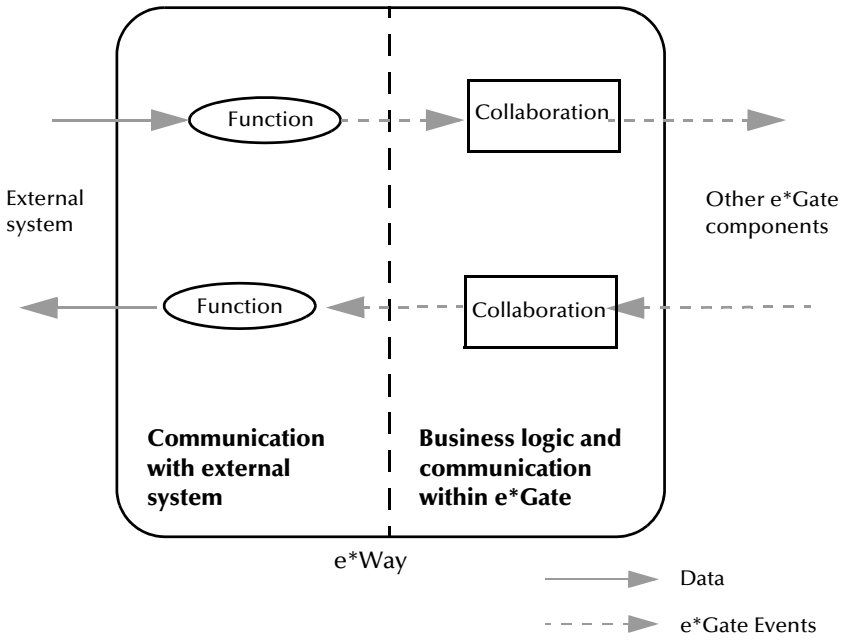
An integer between 1 and 86,400. The default is 10.

3.1.3. Monk Configuration

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

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 2 below) 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 2 e*Way internal architecture



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

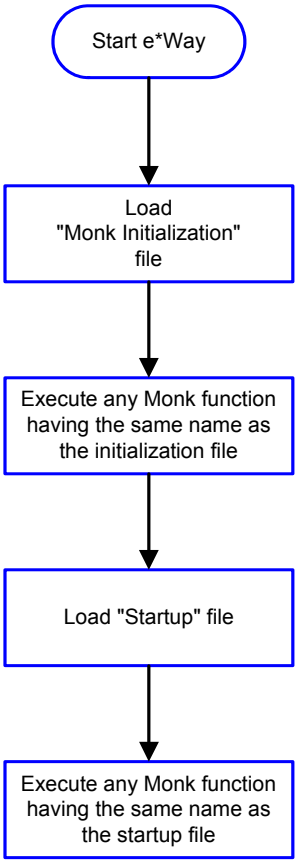
Type of Operation	Name
Initialization	**Startup Function** on page 27 (also see **Monk Environment Initialization File** on page 26)
Connection	**External Connection Establishment Function** on page 29 **External Connection Verification Function** on page 29 **External Connection Shutdown Function** on page 30
Schedule-driven data exchange	**Exchange Data with External Function** on page 28 **Positive Acknowledgment Function** on page 30 **Negative Acknowledgment Function** on page 31
Shutdown	**Shutdown Command Notification Function** on page 31
Event-driven data exchange	**Process Outgoing Message Function** on page 27

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

Initialization Functions

Figure 3 illustrates how the e*Way executes its initialization functions.

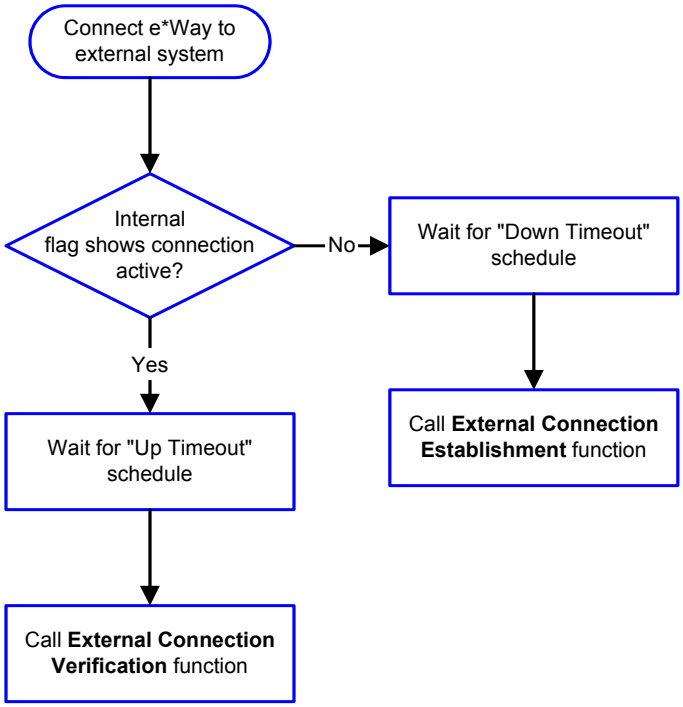
Figure 3 Initialization Functions



Connection Functions

Figure 4 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 4 Connection establishment and verification functions

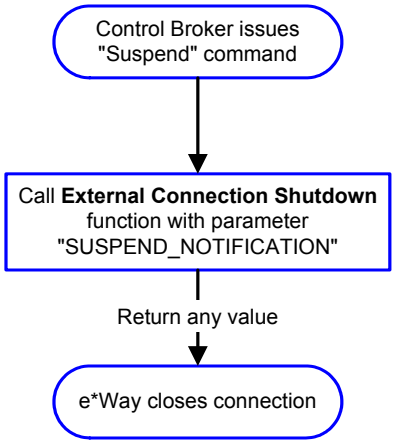


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 6 on page 23** *and* **Figure 8 on page 25** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 50 and* **send-external-down** *on page 49 for more information.*

Figure 5 illustrates how the e*Way executes its "connection shutdown" function.

Figure 5 Connection shutdown function



Schedule-driven Data Exchange Functions

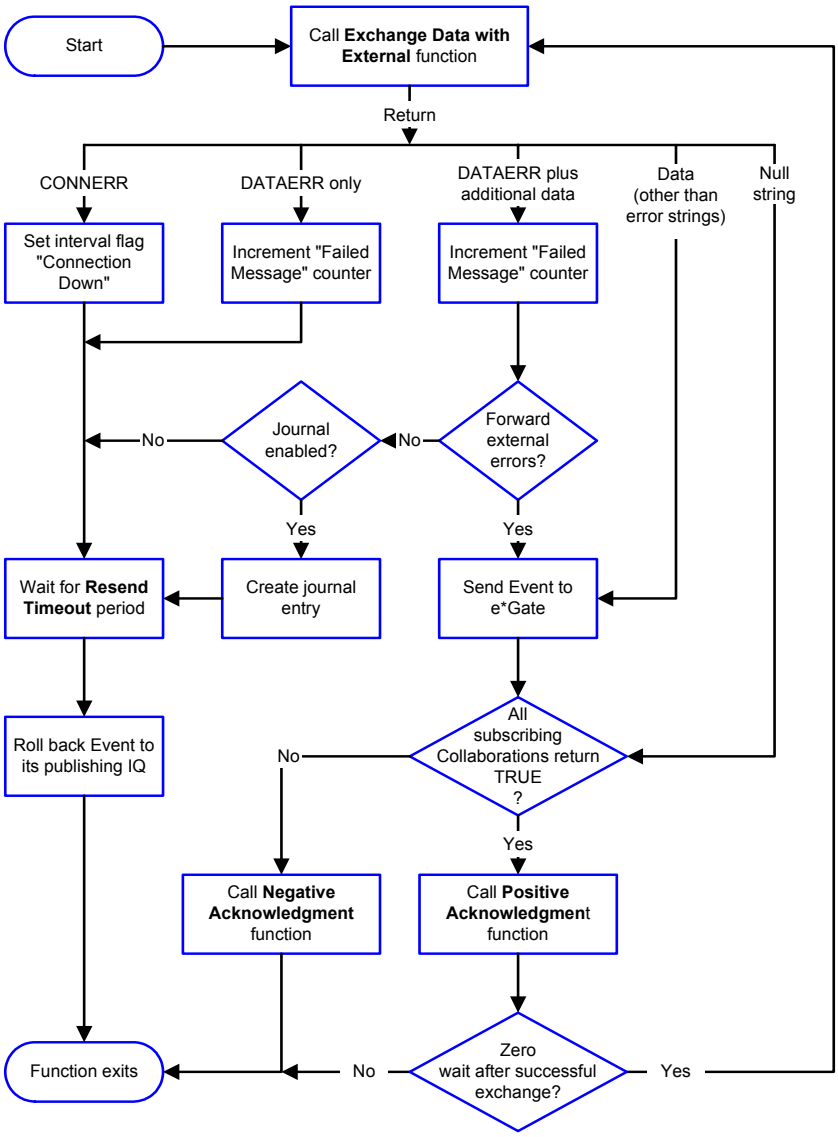
Figure 6 (on the next page) illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgment Function** and **Negative Acknowledgment Function** are also called during this process.

"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 "Stop 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.

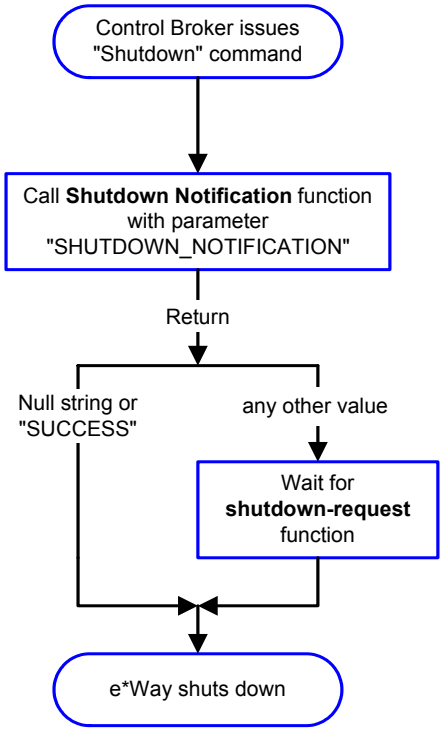
Figure 6 Schedule-driven data exchange functions



Shutdown Functions

Figure 7 illustrates how the e*Way implements the shutdown request function.

Figure 7 Shutdown functions



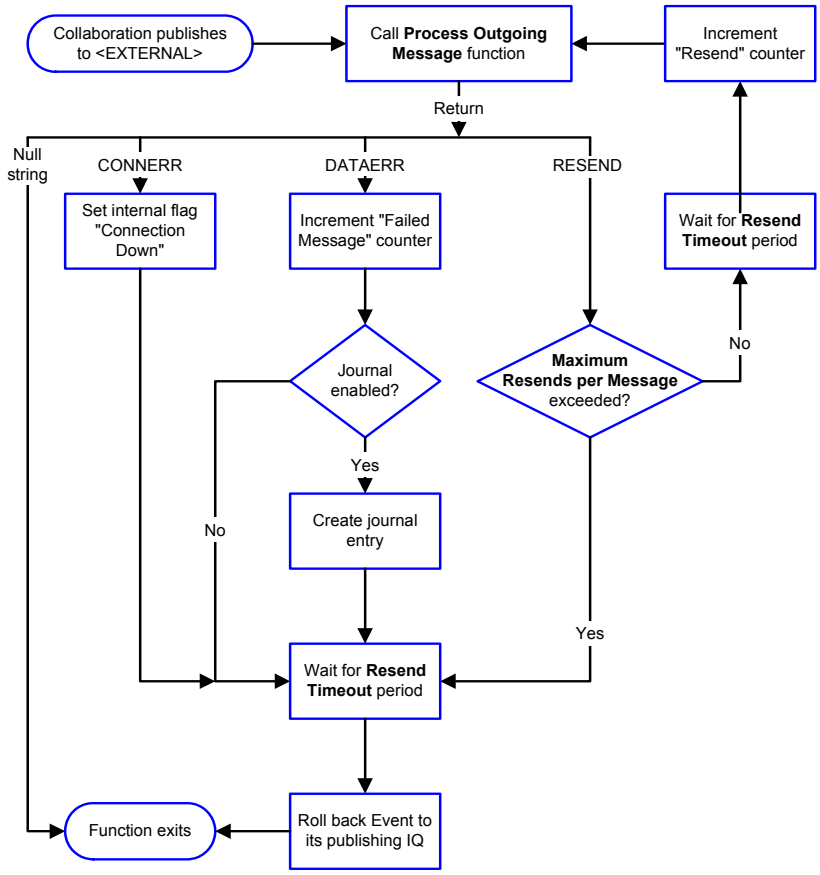
Event-driven Data Exchange Functions

Figure 8 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 8 Event-driven data-exchange functions



How to Specify Function Names or File Names

Parameters that require the name of a Monk function will 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 will be searched after the default load paths.

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.

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 will automatically be loaded into the e*Way's Monk environment. This parameter is optional and may be left blank.

Required Values

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

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.

This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which will be 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 filename within the "load path", or filename plus path information (relative or absolute). If path information is specified, that path will be appended to the "load path." See **"Additional Path" on page 25** for more information about the "load path."

The default is **monk_library/ewcntea/cntea-init.monk**. See **cntea-init** on page 54 for more information.

Additional information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way will load this file and try 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 3 on page 20**).

Startup Function

Description

Specifies a Monk 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 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. The default is **cntea-startup**. See **cntea-startup** on page 57 for more information.

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

The e*Way will load this file and try to invoke a function of the same base name as the file name (see **Figure 3 on page 20**). 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 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 Monk function, or the name of a file (optionally including path information) containing a Monk function. The default is cntea-outgoing. See **cntea-outgoing** on page 56 for more information. ***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 8 on page 25** for more details):

- 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 message (Event) data itself.
- If a string other than the following 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 48 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. The default is cntea-exchange. See **cntea-exchange** on page 53 for more information.

Additional Information

The function accepts no input and must return a string (see **Figure 6 on page 23** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information will be 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 6 on page 23** 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 ACKed or NAKed (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 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 or the schedule is manually invoked using the Monk function **start-schedule** (see **start-schedule** on page 51 for more information).

External Connection Establishment Function

Description

Specifies a Monk 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 Monk function, or the name of a file (optionally including path information) containing a Monk function. The default is **cntea-extconnect**. See **cntea-extconnect** on page 53 for more information. ***This field cannot be left blank.***

Additional Information

The function accepts no input and must return a string:

- "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 will call 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 will execute the **External Connection Establishment** function in its place. The default is cntea-verify. See **cntea-verify** on page 58 for more information.

Additional Information

The function accepts no input and must return a string:

- "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 will call to shut down the connection to the external system.

Required Values

The name of a Monk function. The default is **cntea-shutdown**. See **cntea-shutdown** on page 56 for more information. This parameter is optional.

Additional Information

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

This function will only be invoked when the e*Way receives a "suspend" command from a Control Broker. When the "suspend" command is received, the e*Way will invoke 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 will call 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. The default is **cntea-ack**. See **cntea-ack** on page 52 for more information.

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:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the Positive Acknowledgment function will be 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 will call 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. The default is **cntea-nack**. See **cntea-nack** on page 54 for more information.

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:

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the function will be 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 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 Monk function. The associated function is **cntea-notify**. See **cntea-notify** on page 55 for more information.

Additional Information

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

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

- A null string or "SUCCESS": Indicates that the shutdown can occur immediately.
- Any other string: Indicates that shutdown must be postponed. Once postponed, shutdown will not proceed until the Monk function **shutdown-request** is executed (see **shutdown-request** on page 50).

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.*

3.1.4. Jacada Enterprise/Access Integrator Configuration

The parameters in this section help you set up the required information for the e*Way to access the Jacada Enterprise/Access Integrator application.

Host

Description

Specifies the host on which the Enterprise/Access server is running.

Required Values

A valid host name or IP.

Port

Description

Specifies the port on which the Enterprise/Access server is listening for connections.

Required Values

An integer between 1 and 864000. The default is 30001.

ServiceObject

Description

Specifies the name of the Service Object to which to connect.

Required Values

A string. This value is case sensitive and must match the name as configured in Enterprise Access. This parameter is **mandatory**.

Clientname

Description

Specifies the name of the client system.

Required Values

A string. This value is optional and will appear in the Enterprise Access System Monitor.

Username

Description

Specifies the name of the Enterprise Access System user.

Required Values

A string. This value is optional and will appear in Enterprise Access System Monitor.

Timeout

Description

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

Required Values

An integer between 1 and 864000. The default is 120.

3.2 Environment Configuration

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

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.

Chapter 4

Implementation

This chapter includes information pertinent to implementing the Jacada Enterprise/Access e*Way in a production environment. Also included is a sample output file. This chapter assumes that the Jacada Enterprise/Access e*Way has been successfully installed. (See Chapter 2 for details.)

4.1 Implementation Process: Overview

To implement the Jacada Enterprise/Access e*Way within an e*Gate system, do the following:

- Determine whether or not the client runtime dll, EACIw32.dll has been installed. See **"External System Requirements" on page 8** for more information.
- Define Event Type Definitions (ETDs) to package the data being exchanged with the external system.
- In the e*Gate Schema Designer, do the following:
  - Define Collaboration Rules to process Event data.
  - Define any IQs to which Event data will be published prior to sending it to the external system.
  - Define the e*Way component (this procedure is discussed in **Chapter 2**).
  - Within the e*Way component, configure Collaborations to apply the required Collaboration Rules.
- Use the e*Way Editor to set the e*Way's configuration parameters.
- Be sure that any other e*Gate components are configured as necessary to complete the schema.
- Test the schema and make any necessary corrections. See **"Sample Output File" on page 47** for examples of how the above steps are combined to create a working implementation.

Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's Help system.*

4.2 Creating Event Type Definitions from Sample Data

You can use the ETD Editor to create or modify any necessary Event Type Definitions using the Jacada Converter. The Converter automatically creates an Event Type Definition file based upon sample data.

There are two ways to launch the Jacada Converter: from the command line and from the ETD Editor.

4.2.1. Creating Event Type Definitions using Command-line Utilities

To create an ETD using the Jacada Converter command-line utility:

From the command line, type the following on one line:

```
stccnteaconvert.exe [options] [@commandfile] <filename>
```

where:

Command	Description
-h	This screen
-v	Verbose
--ver	Show version
-d	Use log file
-rh <param>	Registry host
-rs <param>	Registry schema name
-rp <param>	Registry port
-up <param>	Password
-tf <param>	Temporary return filename
-eh <param>	Jacada Integrator Environment Manager Host
-ep <param>	Jacada Integrator Environment Manager Port (default = 30001)
-cl <param>	Client Name (optional)
-un <param>	UserName (optional)
-to <param>	Default Timeout In Seconds (default =120)
-obj <param>	Service Object (case sensitive)
-del <param>	Delimiters (default "|~^$#*" in the order listed)
-out <param>	Output SSC filename

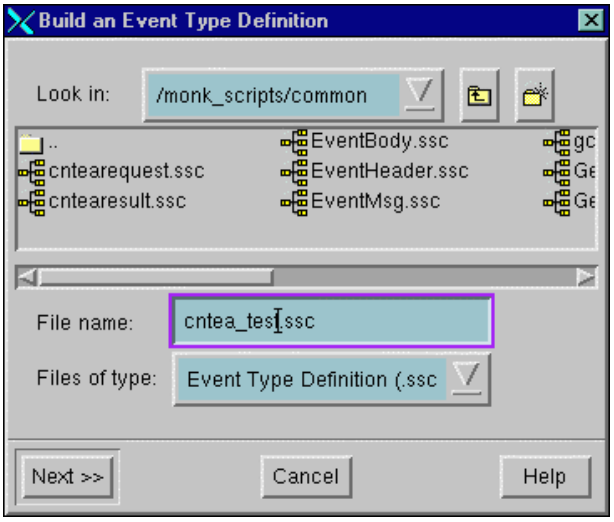
4.2.2. Creating Event Type Definitions from the ETD Editor

To create an ETD using the Jacada Converter from the ETD Editor:

1 Launch the ETD Editor.

2 On the ETD Editor's Toolbar, click **Build**. The **Build an Event Type Definition** dialog box appears.

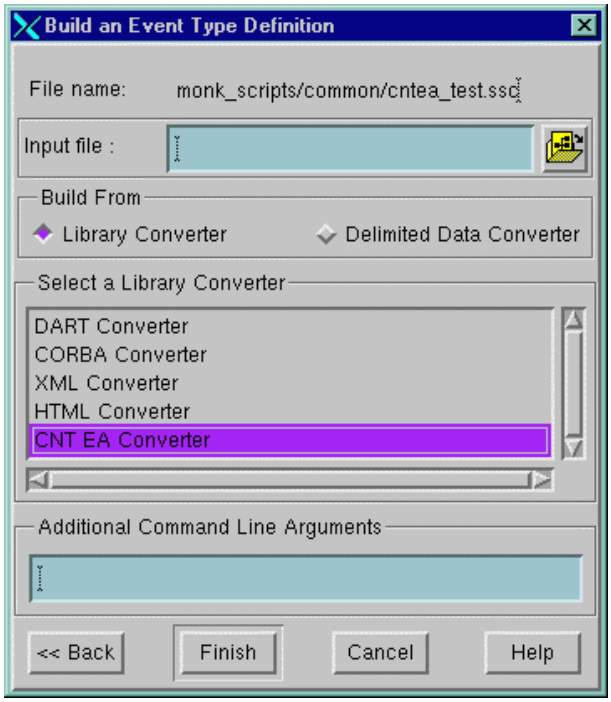
Figure 9 Build an Event Type Definition



3 In the **File name** field, type the name of the ETD file you wish to build. Do not specify any file extension--the Editor will supply the .ssc extension automatically.

4 Click **Next**. A new dialog box appears.

Figure 10 Build an Event Type Definition - Jacada Converter



5 If the Converter requires a data file, enter the name of the input file on which to base the ETD in the **Input file** box.

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

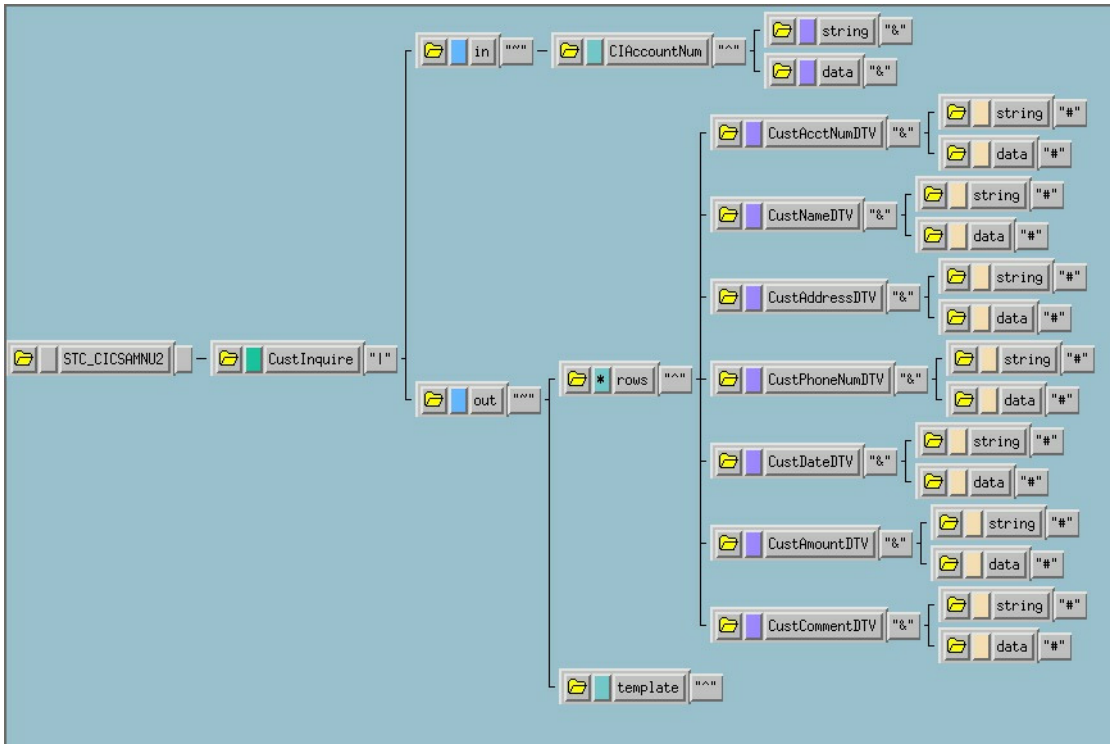
7 Under **Select a Library Converter**, select **Jacada Converter**.

8 Check the Configuration to determine any required parameters in the Service Object. Enter these parameters in the Additional command-line arguments box.

9 Click **Finish**. The Build tool will create the ETD.

Note: *The STC_CICSAMNU2 Service Object was built at SeeBeyond for testing purposes. Although these exact collaborations cannot be used, the concepts are the same for your instance of the service object you create. See Figure 11 to view the ETD.*

Figure 11 STC_CICSAMNU2 Event Type Definition



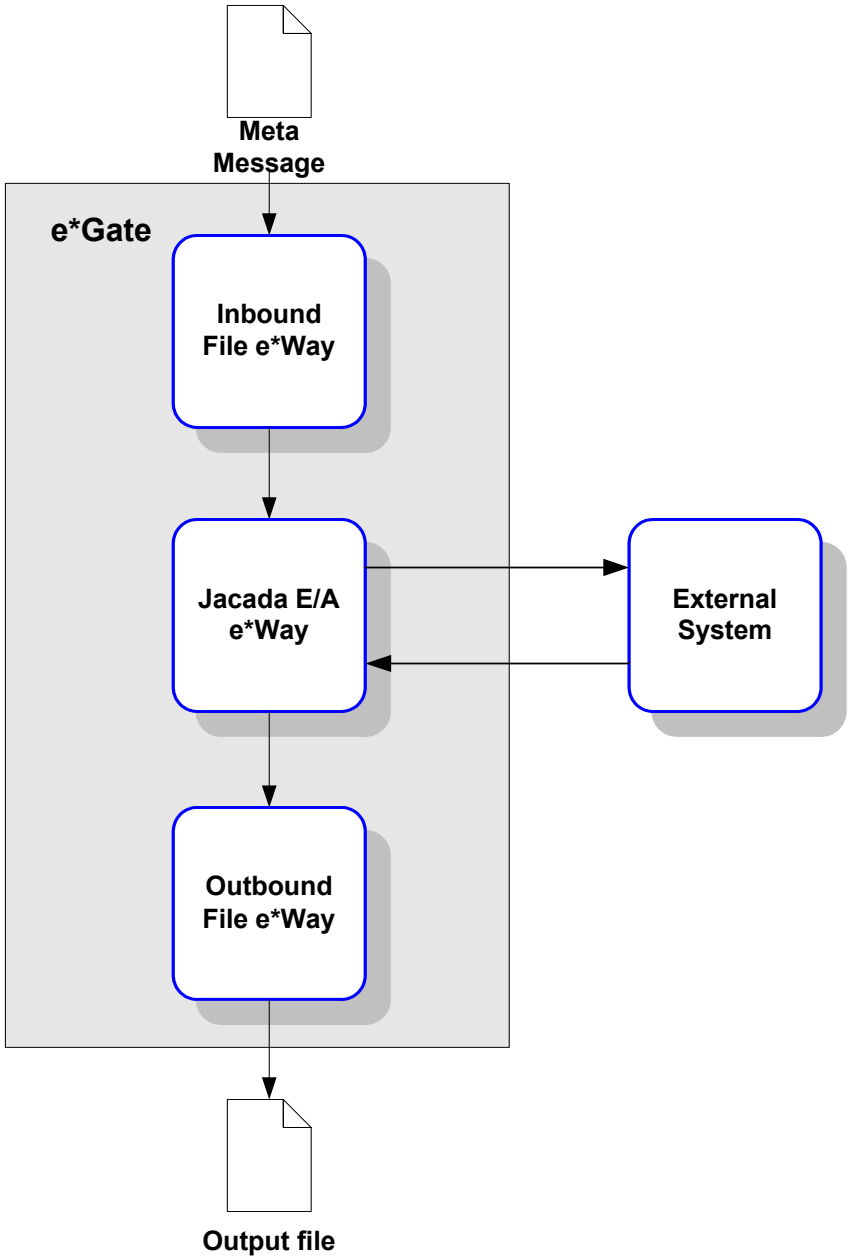
4.3 Sample Configurations

This section describes how to use the Jacada Enterprise/Access e*Way with a sample configuration. This sample will read an input file, apply associated Collaborations and Rules, and then output the results in a specified location. In order to successfully produce the output, the name of the Service Object (specified in the Jacada Enterprise/Access Integrator Configuration) and the names of the methods and parameters in the Service Object must be identified.

The name of the Service Object, in this example, is **STC_CICSAMNU2**. The Methods used are:

- CustUpdate
- CustBrowse
- CustInquire
- CustAdd

Figure 12 Sample schema basic architecture

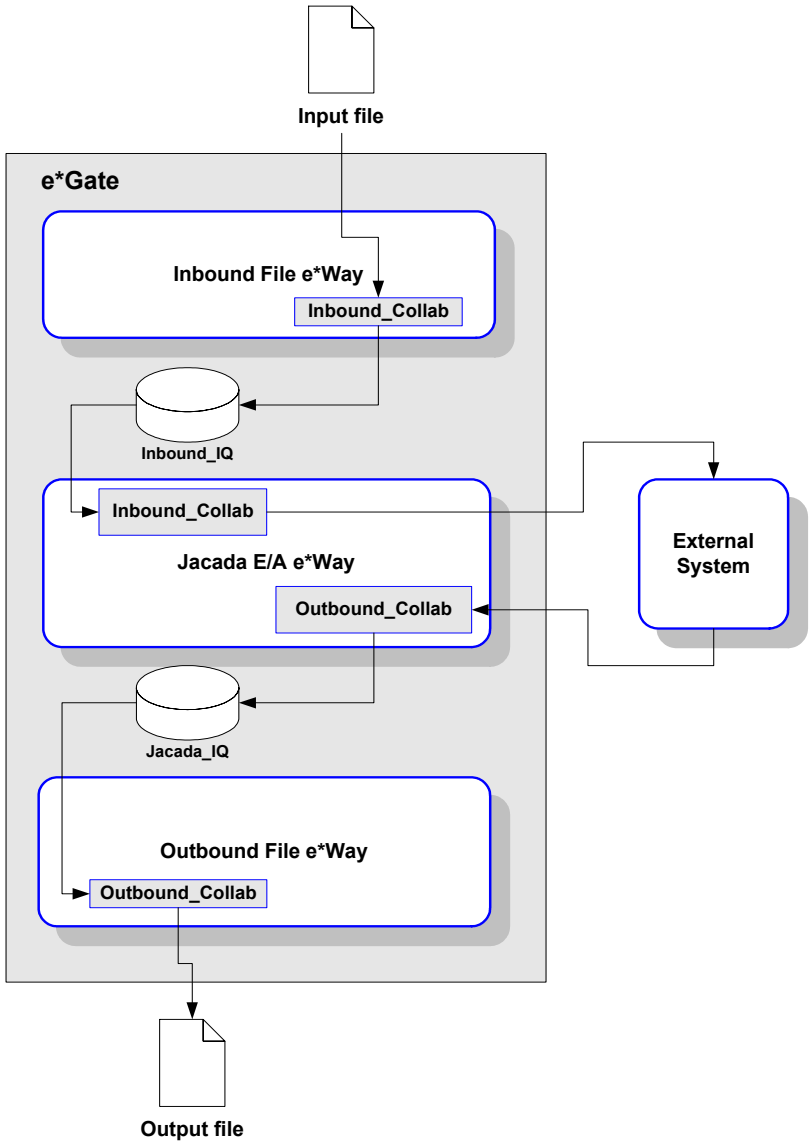


e*Gate initiates the session with the external system after creating a Meta Data Structure (Event) and sending it to the Jacada Enterprise/Access e*Way as a trigger. This Event contains all the information necessary for the e*Way to communicate with the external system. This information can be used to initiate, respond to, or validate the data interchange.

The Jacada Enterprise/Access e*Way provides an **invoke** function that takes in methods and parameters which in turn enable interaction with the E/A Service Object.

This schema requires a number of components, as illustrated in Figure 13.

Figure 13 Sample schema (component view)



Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's Help system.*

4.3.1. Creating a schema using the STC_CICSAMNU2 Service Object

Note: *The STC_CICSAMNU2 Service Object was created at SeeBeyond for testing purposes. Although these exact collaborations cannot be used, the concepts are the same for the instance of the service object you create*

1. Log into the e*Gate Schema Designer and click **New** to create a new schema. Name the schema "Jacada_Sample1."

   The Schema Designer main screen appears.
2. If the Navigator's **Components** tab is not selected already, select it now.
3. Create an Event Type named "Event1."

Note: *The Jacada Enterprise/Access Integrator includes two pre-defined Monk Script files,* ***cntearequest.ssc*** *and* ***gcntearesult.ssc****. Use one of these scripts when defining Event 1.*

4. Display the properties of the **Event1** Event Type. Then, use the **Find** button, select **cntrequest.ssc** to assign the file to the **Event1** Event Type.
5. Create a Collaboration Rule named "Jacada_Collab."
6. Edit the Properties of this Collaboration Rule as follows:

Service	Pass Through
Subscription	**Event1** (the Event Type defined in Step 3 above)
Publication	**Event1** (Event Type defined in Step 3 above)

7. Create two IQs, named "Inbound_IQ" and "Jacada_IQ."
8. Create an Input File, using an ASCII text editor. The input must have the following format:

   ```
   ServiceObjectName|Method*Parameter^Value*Parameter^Value
   ```
9. Create an e*Way named "Inbound."
10. Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewfile.exe**.

The next part of the procedure requires that you launch the e*Way Editor and define the file-based e*Way's properties.

1 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

2 Using the e*Way Editor, make the following configuration settings:

Section	Parameter and setting
General Settings	**AllowIncoming**:Yes **AllowOutgoing**: No
Poller (inbound) Settings	**Polldirectory**: Set to the same location in which the Input File created in Step 9 above resides. **Input File Mask**: Set to the same extension as that of the Input File created in Step 9 above.

3 Save the settings, promote to run time and exit the e*Way Editor.

4 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create a Collaboration for the Inbound e*Way.

1 Open the **Inbound** e*Way and create a Collaboration named "Inbound_Collab."

2 Set the Collaboration's properties as follows:

Collaboration Rule	**Jacada_Collab**
Subscriptions	Event: **Event1** Source: **<External>**.
Publications	Event: **Event1** Publish to: **Inbound_IQ**.

Now that the "inbound" e*Way is completely configured, you must create an e*Way that communicates with the outbound file e*Way.

1 Create a new e*Way component named "Jacada_eway."

2 Display the e*Way's properties. Then, use the **Find** button to assign the file **stcewgenericmonk.exe**.

3 Click **New** to launch the e*Way Editor. When prompted with a list of templates, select **stcewcntea**.

4 Use the e*Way Editor to define the following parameters:

Section	Parameter and Settings
General Settings	Leave all settings unchanged
Communication Setup	**Exchange Data Interval**: 120 (default) **Zero Wait Between Successful Exchanges**: No (default) **Start Exchange Data Schedule**: (unchanged) **Stop Exchange Data Schedule**: (unchanged) **Down Timeout**: 15 (default) **Up Timeout**: 15 (default) **Resend Timeout**: 10 (default)
Monk Configuration	**Additional Path**: (unchanged) **Auxiliary Library Directories**: monk_library/ewcntea **Monk Environment Initialization File**: monk_library/ewcntea-init.monk **Startup Function**: cntea-startup **Process Outgoing Message Function**: monk_library/ewcntea/stccicsamnu-pomf.tsc **Exchange Data With External Function**: cntea-exchange **External Connection Establishment Function**: cntea-connect **External Connection Verification Function**: cntea-verify **External Connection Shutdown Function**: cntea-shutdown **Positive Acknowledgment Function**: cntea-ack **Negative Acknowledgment Function**: cntea-nack **Shutdown Command Notification Function**: (unchanged) The remaining parameters may be left blank for this sample.
Jacada Configuration	**Host**: host name/ip where the Jacada Enterprise/Access server is running **Port**: 30001 **Service Object**: Your Service Object (STC_CICSAMNU2 used for this test) **Clientname**: Enter an appropriate client name (if necessary). **Username**: Enter an appropriate user name if necessary. **Timeout**: 120 (or higher)

5 Save the settings and exit the e*Way Editor. Promote to Runtime.

6 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

Next, create the first Collaboration for the Jacada Enterprise/Access e*Way.

1 Select the **Jacada_eway** component and create a Collaboration named "IN_Collab."

2 Assign the following properties to the Collaboration:

Collaboration Rules	Jacada_Collab
Subscriptions	Event: Event1 Source: External
Publications	Event: Event1 Publish to: Jacada_IQ

Create the second Collaboration for the Jacada Enterprise/Access e*Way.

1 Select the Jacada_eway component and create a Collaboration named "OUT_Collab."

2 Assign the following properties to the Collaboration:

Collaboration Rules	Jacada_Collab
Subscriptions	Event: Event1 Source: Inbound_Collab
Publications	Event: Event1 Publish to: <External>

3 Save the settings and exit the e*Way Editor.

Next, create and configure the final e*Way component.

1 Create a new e*Way component named "Outbound."

2 In its Properties page, specify the executable file of "Outbound" as **stcewfile.exe**.

3 Display the e*Way's properties. Then, use the **Find** button, navigate to the "bin" folder to assign the file **stcewfile.exe**.

4 With the e*Way's Properties page still displayed, click **New** to launch the e*Way Editor.

5 Using the e*Way Editor, configure the following settings:

Section	Parameter and Settings
General Settings	**Allow Incoming**: No **Allow Outgoing**: Yes
Poller (inbound) Settings	**Poll Directory**: Set to the same location in which the Input File created in Step 9 above resides. **Input File Mask**: Specify the name of the output file to produce. For example, **Jacada_out.txt**.

6 Save the settings, promote to run time and exit the e*Way Editor.

7 When you return to the e*Way's Properties page, click **OK** to save all changes and return to the Schema Designer's main window.

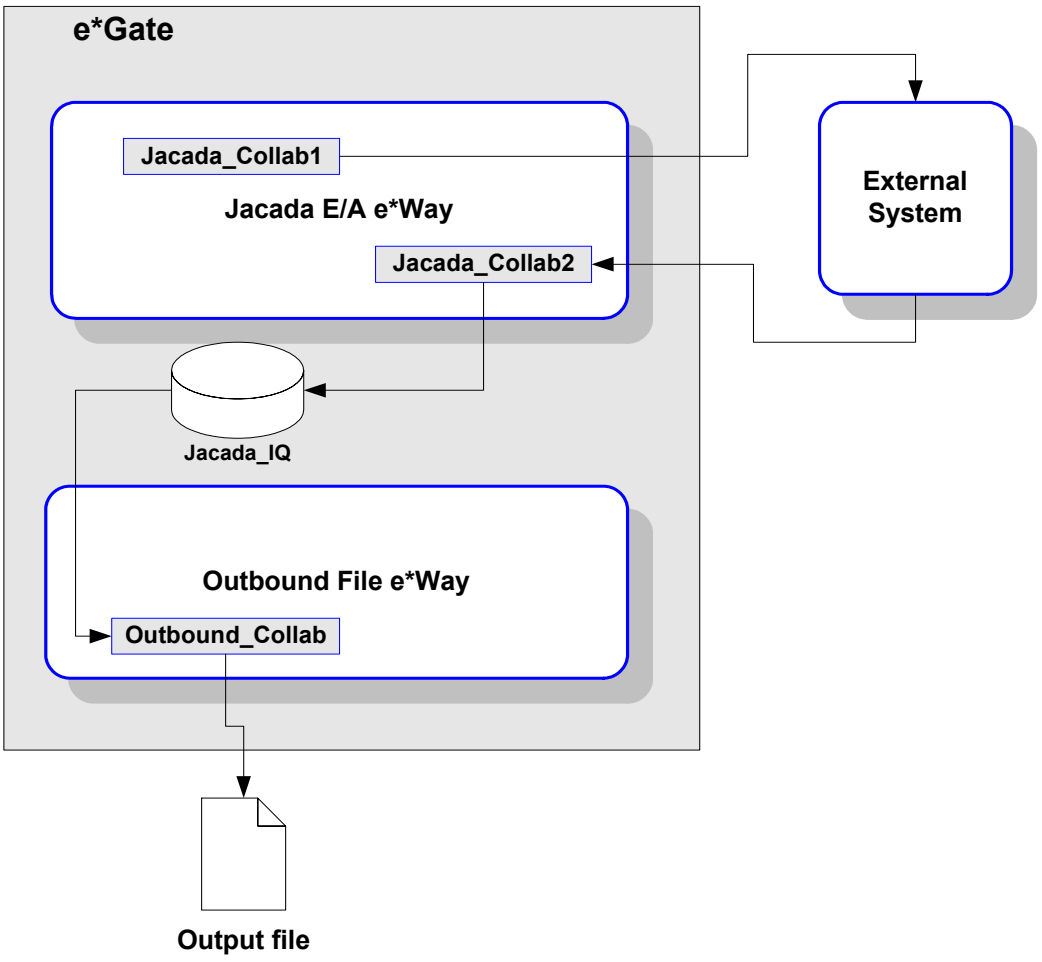
Next, create the Collaboration for the Outbound e*Way.

1 Select the **Outbound e*Way** component and create a Collaboration named "Outbound_Collab."

2 Assign the following properties to the Collaboration:

Collaboration Rules	Jacada_Collab
Subscriptions	Event: Event1 Source: In_Collab
Publications	Event: Event1 Publish to: <External>

Figure 14 Sample Jacada Enterprise/Access schema



Note: *For more information about creating or modifying any component within the e*Gate Schema Designer, see the Schema Designer's Help system.*

The Schema Designer configuration is now complete. Now, you must create some test data which will be sent via the Jacada Enterprise/Access e*Way to an external system. The results of these requests will be saved to the output data file.

Note: *This sample schema will only work with the STC_CICSAMNU2 Service Object. The schema will not run successfully without it. This sample configuration is provided for demonstration purposes only.*

4.4 Sample Monk Scripts

The samples in this section can be run using the **stctrans** command-line utility. They do not require a complete e*Gate schema configuration to function, and are designed to illustrate the principles involved in creating your own custom Monk scripts. The library (dll) files to be loaded and the script to be tested must be in the load path (or, for simplicity's sake, may be placed in the connected directory). See the *Monk Developer's Reference* for more information about the load path.

The syntax of the **stctrans** utility is

```
stctrans monk_file.monk
```

Additional command-line flags are available; enter **stctrans -h** to display a list, or see the *e*Gate Integrator System Administration and Operations Guide* for more information.

The sample files may be created using any text editor. The samples use a generic "**www.abc.com**" site name; before testing any script, replace the generic name with a working site name.

4.4.1. Implementing the Sample Configuration

This sample will essentially generate a simple message, and will serve to verify that the Jacada Enterprise/Access e*Way has been properly installed and configured. The following steps provide the specifics for implementing the sample configuration.

4.4.2. Expected Results of Sample Configuration

The CustBrowse and CustInquire methods for the STC_CICSAMNU2 Service Object each produce an output file.

For example, the sample input:

```
STC_CICSAMNU2|CustBrowse*CBAccountNum^000001
```

browses the records on a database using a specific parameter, and produces an output file in the directory specified by the Jacada Enterprise/Access e*Way configuration file settings.

Another example,

```
STC_CICSAMNU2|CustInquire*CIAccountNum^000102
```

requests the record of a single customer, and produces an output file containing the customer records retrieved.

Alternatively, the following sample input

```
STC_CICSAMNU2|CustAdd*CAAccountNum^000224*CACustName^Name*CACustAddre
ss^Address*CACustPhoneNum^###-####*CACustDate^mm-dd-
yy*CACustAmount^####*CACustComment^Comment
```

adds a new record for a customer into the database, but does not produce any output file. However, the success of this sample input may be checked by using the CustInquire method on the customer that was just added and inserting the customer's account number into the AccountNum field. (The sample input must all be on 1 line)

"The following sample input

```
STC_CICSAMNU2|CustUpdate*CUAccountNum^000224*CUCustName^Name*CUCustAd
dress^Address*CU     CustPhoneNum^###-####*CUCustDate^mm-dd-
yy*CUCustAmount^####*CUCustComment^Comment
```

updates an existing record for a customer into the database, but does not produce any output file. The success of the input may be checked by using the CustInquire method on the customer whose record was just modified."

Note: *You will need to create input files that will work with your service object in order to test yourschema configuration.*

Sample Output File

```
Result Row [0]=AcctNumTTV^000100 *AmountTTV^$0100.11 *CustNameTTV^S.
D. BORMAN        *
   Attribute = AcctNumTTV   Value = 000100
   Attribute = AmountTTV   Value = $0100.11
   Attribute = CustNameTTV   Value = S. D. BORMAN
Result Row [1]=AcctNumTTV^000102 *AmountTTV^$1111.11 *CustNameTTV^J.
T. CZAYKOWSKI    *
   Attribute = AcctNumTTV   Value = 000102
   Attribute = AmountTTV   Value = $1111.11
   Attribute = CustNameTTV   Value = J. T. CZAYKOWSKI
Result Row [2]=AcctNumTTV^000104 *AmountTTV^$0999.99 *CustNameTTV^M.
B. DOMBEY        *
   Attribute = AcctNumTTV   Value = 000104
   Attribute = AmountTTV   Value = $0999.99
   Attribute = CustNameTTV   Value = M. B. DOMBEY
Result Row [3]=AcctNumTTV^000106 *AmountTTV^$0087.71 *CustNameTTV^A.
I. HICKSON       *
   Attribute = AcctNumTTV   Value = 000106
   Attribute = AmountTTV   Value = $0087.71
   Attribute = CustNameTTV   Value = A. I. HICKSON
Result Row [4]=AcctNumTTV^000111 *AmountTTV^$0111.11
*CustNameTTV^ALAN TULIP          *
   Attribute = AcctNumTTV   Value = 000111
   Attribute = AmountTTV   Value = $0111.11
   Attribute = CustNameTTV   Value = ALAN TULIP
Result Row [5]=AcctNumTTV^000762 *AmountTTV^$0000.00
*CustNameTTV^SUSAN MALAIKA       *
   Attribute = AcctNumTTV   Value = 000762
   Attribute = AmountTTV   Value = $0000.00
   Attribute = CustNameTTV   Value = SUSAN MALAIKA
Result Row [6]=AcctNumTTV^000983 *AmountTTV^$9999.99 *CustNameTTV^J.
S. TILLING       *
   Attribute = AcctNumTTV   Value = 000983
   Attribute = AmountTTV   Value = $9999.99
   Attribute = CustNameTTV   Value = J. S. TILLING
Result Row [7]=AcctNumTTV^001222 *AmountTTV^$3349.99
*CustNameTTV^D.J.VOWLES          *
   Attribute = AcctNumTTV   Value = 001222
   Attribute = AmountTTV   Value = $3349.99
   Attribute = CustNameTTV   Value = D.J.VOWLES
Result Row [8]=AcctNumTTV^001781 *AmountTTV^$0009.99
*CustNameTTV^TINA J YOUNG        *
   Attribute = AcctNumTTV   Value = 001781
   Attribute = AmountTTV   Value = $0009.99
   Attribute = CustNameTTV   Value = TINA J YOUNG
Result Row [9]=AcctNumTTV^003210 *AmountTTV^$3349.99
*CustNameTTV^B.A. WALKER         *
   Attribute = AcctNumTTV   Value = 003210
   Attribute = AmountTTV   Value = $3349.99
   Attribute = CustNameTTV   Value = B.A. WALKER
Result Row [10]=AcctNumTTV^003214 *AmountTTV^$0009.99
*CustNameTTV^PHIL CONWAY         *
   Attribute = AcctNumTTV   Value = 003214
   Attribute = AmountTTV   Value = $0009.99
   Attribute = CustNameTTV   Value = PHIL CONWAY
Result Row [11]=AcctNumTTV^003890 *AmountTTV^$0009.99
*CustNameTTV^BRIAN HARDER        *
   Attribute = AcctNumTTV   Value = 003890
   Attribute = AmountTTV   Value = $0009.99
   Attribute = CustNameTTV   Value = BRIAN HARDER
```

Chapter 5

Jacada Enterprise/Access e*Way Functions

The Jacada Enterprise/Access e*Way's functions fall into the following categories:

5.1 Basic Functions

The functions described in this section 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.

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

The basic functions are

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

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 Schema Designer).

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, in invokes the Shutdown Command Notification Function (see**"Shutdown Command Notification Function" on page 31**). 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 Data 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 Data with External" function specified within the e*Way's configuration file. Execution will be 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 Jacada Enterprise/Access e*Way Standard Functions

The functions in this section are accessed via the e*Way's communications setup and are defined within the configuration file.

The standard functions are:

cntea-ack

Syntax

```
(cntea-ack arg)
```

Description

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

Parameters

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

Return Values

string

An empty string indicates a successful operation. The e*Way will then be able to proceed with the next request.

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

Additional Information

See **"Positive Acknowledgment Function" on page 30** for more information.

cntea-exchange

Syntax

```
(cntea-exchange)
```

Description

cntea-exchange sends a received Event from the external system to e*Gate. The function expects no input.

Parameters

None.

Return Values

String

An empty string indicates a successful operation. Nothing is sent to e*Gate.

A string, containing Event data, indicates successful operation, and the returned Event is sent to e*Gate.

"CONNERR" indicates a problem with the connection to the external system. When the connection is re-established this function will be re-executed with the same input Event.

Throws

None.

Additional Information

See **"Exchange Data with External Function" on page 28** for more information.

cntea-extconnect

Syntax

```
(cntea-extconnect)
```

Description

cntea-extconnect establishes a connection to the external system.

Parameters

None.

Return Values

string

"UP" indicates the connection is established. Anything else indicates no connection.

Throws

None.

Additional Information

See **"External Connection Establishment Function" on page 29** for more information.

cntea-init

Syntax

```
(cntea-init)
```

Description

cntea-init begins the initialization process for the e*Way. This function loads the **stc_monkcntea.dll** file and the initialization file, thereby making the function scripts available for future use.

Parameters

None.

Return Values

string
If a "FAILURE" string is returned, the e*Way will shut down. Any other return indicates success.

Throws

None.

Additional Information

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

See **"Monk Environment Initialization File" on page 26**

cntea-nack

Syntax

```
(cntea-nack arg)
```

Description

cntea-nack sends a negative acknowledgment to the external system when the e*Way fails to process and queue Events from the external system.

Parameters

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

Return Values

string
An empty string indicates a successful operation.

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

Throws

None.

Additional Information

See **"Negative Acknowledgment Function" on page 31** for more information.

cntea-notify

Syntax

```
(cntea-notify command)
```

Description

cntea-notify notifies the external system that the e*Way is shutting down.

Parameters

Name	Type	Description
command	string	When the e*Way calls this function, it will pass the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

string
Returns a null string.

Throws

None.

Additional Information

See **Shutdown Command Notification Function** on page 31 for more information.

cntea-outgoing

Syntax

```
(cntea-outgoing event-string)
```

Description

cntea-outgoing is used for sending a received message from e*Gate to the external system.

Parameters

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

Return Values

string

An empty string indicates a successful operation.

"RESEND" causes the Event to be immediately resent.

"CONNERR" indicates a problem with the connection to the external system. When the connection is re-established this function will be re-executed with the same input Event.

"DATAERR" indicates the function had a problem processing data. If the e*Gate journal is enabled, the Event is journaled and the failed Event count is increased. (The input Event is essentially skipped in this process.) Use the **event-send-to-egate** function to place failed events in a failed Event queue. See **event-send-to-egate** on page 48 for more information on this function.

Additional Information

See **Process Outgoing Message Function** on page 27 for more information.

cntea-shutdown

Syntax

```
(cntea-shutdown shutdown)
```

Description

cntea-shutdown requests that the external connection shut down. A return value of "SUCCESS" indicates that the shutdown can occur immediately. Any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a **shutdown-request** (See **shutdown-request** on page 50 for more information) call from within a Monk function to allow the requested shutdown to continue.

Parameters

Name	Type	Description
shutdown	string	The string "SUSPEND_NOTIFICATION" is passed as the parameter when the e*Way calls this function.

Return Values

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

Throws

None.

Additional Information

See **"External Connection Shutdown Function" on page 30** for more information.

cntea-startup

Syntax

```
(cntea-startup)
```

Description

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

Parameters

None.

Return Values

string
"FAILURE" causes shutdown of the e*Way. Any other return indicates success.

Throws

None.

Additional Information

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

See **"Startup Function" on page 27** for more information.

cntea-verify

Syntax

```
(cntea-verify)
```

Description

cntea-verify is used to verify whether the connection to the external system is established.

Parameters

None.

Return Values

string
Returns "UP" or "SUCCESS" if the connection is established. Any other value indicates the connection is not established.

Throws

None.

Additional Information

See **"External Connection Verification Function" on page 29** for more information.

5.3 Jacada Enterprise/Access Native Functions

The native Jacada Enterprise/Access functions are used to establish and direct contact with the external host system.

The functions described in this section can only be called from within a Collaboration Rules script.

The Jacada Enterprise/Access native functions are:

cntea-call-struct

Syntax

```
(cntea-call-struct svcobjecthandle svcobjectpath methodpath)
```

Description

cntea-call-struct traverses the input argument nodes and their data nodes in order to set the appropriate input arguments to a Jacada Enterprise/Access method invocation.

Parameters

Name	Type	Description
svcobjecthandle	opaque handle	The handle to the service object created.
svcobjectpath	string	The path to the service object node (first node) in the ETD. For example: **~output%STC_CICSAMNU2**
methodpath	string	The path to the method node in the ETD. For example: **~output%STC_CICSAMNU2.CustInquire**

Return Values

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

Throws

None.

Additional Information

The code provided is generic enough to traverse any method-path of a service object Event Type Definition generated by the Jacada build tool (**stccnteaconvert.exe**). Once the input arguments are set, this function invokes the method. If data is returned, the function populates the appropriate data subnodes of the rows node with the returned data from Jacada Integrator. The rows node is a subnode of the out node for the method (see the out node for method **CustInquire** in the sample ETD).

The data node of one or more input arguments can be populated or left blank before this function is called. For example, the data node for **CIAccountNum** is populated with **000100** as the "Account Number" to query.

The caller of this function should check the return code. If the method returns data (the out node has subnodes), the data nodes of the output parameters of the rows node will contain the returned data from Jacada Enterprise/Access Integrator. For example, on return from invoking the CustInquire method, the data node for CustNameDTV is "John Doe".

There are four sample collaborations that use this function included on the CD.

Note: *These sample collaborations use the stc_cicsamnu2 ETD, which was built by the stccnteaconvert.exe build tool. The STC_CICSAMNU2 service object was built by SeeBeyond to demonstrate concepts in documentation form. Although these exact collaborations can not be used, the concepts are the same for your instance of the service object built on your site.*

These collaborations are included for a reference. Sample input data files for these collaborations are also included.

cntea-connect

Syntax

```
(cntea-connect pszHostName dwPort secDefaultTimeout pszClientName
pszUserName)
```

Description

cntea-connect establishes the connection with the specified host.

Parameters

Name	Type	Description
pszHostName	string	The hostname for the connection.
dwPort	integer	The port location for the connection.
secDefaultTimeout	integer	The default timeout, in seconds, for other operations on this connection.
pszClientName	string	The client (machine) name.
pszUserName	string	The user name.

Return Values

handle
The handle associated with the session.

Throws

None.

cntea-disconnect

Syntax

```
(cntea-disconnect hcon)
```

Description

cntea-disconnect closes the connection to the host.

Parameters

Name	Type	Description
hcon	opaque handle	The handle returned by **cntea-open**.

Return Values

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

Throws

None.

cntea-isconnected

Syntax

```
(cntea-isconnected hcon)
```

Description

cntea-isconnected checks to see if the server is up and the connection open.

Parameters

Name	Type	Description
hcon	opaque handle	The handle returned by **cntea-open**.

Return Values

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

Throws

None.

cntea-open-so

Syntax

```
(cntea-open-so hcon pszServObjName)
```

Description

cntea-open-so opens a service object.

Parameters

Name	Type	Description
hcon	opaque handle	The handle returned by **cntea-open**.
pszServObjName	string	The name of the service object.

Return Values

handle
The handle associated with the service object.

Throws

None.

cntea-so-isvalid

Syntax

```
(cntea-so-isvalid hservobj)
```

Description

cntea-so-isvalid checks the validity of the service object handle.

Parameters

Name	Type	Description
hservobj	opaque handle	The handle returned by **cntea-open-so**.

Return Values

Boolean
Returns **#t** (true) if the service object handle is valid; otherwise, returns **#f** (false).

Throws

None.

cntea-close-so

Syntax

```
(cntea-close-so hservobj)
```

Description

cntea-close-so closes the specified service object.

Parameters

Name	Type	Description
hservobj	opaque handle	The handle returned by **cntea-open-so**.

Return Values

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

Throws

None.

Additional Information

This function **must** be called to destroy the handle and deallocate the memory associated with the handle returned by **cntea-open-so**, even if **cntea-so-isvalid** returns **#f**.

cntea-get-method-count

Syntax

```
(cntea-get-method-count hservobj)
```

Description

cntea-get-method-count obtains a count of the methods supported by a service object.

Parameters

Name	Type	Description
hservobj	opaque handle	The handle returned by **cntea-open-so**.

Return Values

integer
Returns the number of methods the service object supports.

Throws

None.

cntea-get-next-method-name

Syntax

```
(cntea-get-next-method-name hservobj)
```

Description

cntea-get-next-method-name obtains the next method name in the list of methods.

Parameters

Name	Type	Description
hservobj	opaque handle	The handle returned by **cntea-open-so**.

Return Values

string
Returns the next method name of the service object. Returns an empty string if there are no more methods to return.

Throws

None.

Additional Information

1. Call **cntea-get-method-count** to get the count of methods.
2. Call **cntea-next-method-name** in a loop to iterate through and retrieve the names of all the methods.

cntea-query-method

Syntax

```
(cntea-query-method hservobj pszMethodName)
```

Description

cntea-query-method queries a specified service object to see if a particular method is supported by the service object.

Parameters

Name	Type	Description
hservobj	opaque handle	The handle returned by **cntea-open-so**.
pszMethodName	string	The name of the method to query.

Return Values

Boolean
Returns **#t** (true) if the method is supported; otherwise, returns **#f** (false).

Throws

None.

cntea-create-method

Syntax

```
(cntea-create-method hservobj pszMethodName)
```

Description

cntea-create-method creates a service object method.

Parameters

Name	Type	Description
hservobj	opaque handle	The handle returned by **cntea-open-so**.
pszMethodName	string	The name of the method supported by the service object.

Return Values

handle
The handle associated with the service object method created.

Throws

None.

cntea-method-isvalid

Syntax

```
(cntea-method-isvalid hmethod)
```

Description

cntea-method-isvalid checks the validity of the service object method handle.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

Boolean
Returns **#t** (true) if the service object method handle is valid; otherwise, returns **#f** (false).

Throws

None.

cntea-destroy-method

Syntax

```
(cntea-destroy-method hmethod)
```

Description

cntea-destroy-method deallocates the memory associated with the service object method previously created by **cntea-create-method**.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-destroy-method**.

Return Values

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

Throws

None.

Additional Information

This function must be called to destroy the handle and deallocate the memory associated with the handle returned by **cntea-create-method**, even if **cntea-method-isvalid** returns **#f**.

cntea-get-inparam-count

Syntax

```
(cntea-get-inparam-count hmethod)
```

Description

cntea-get-inparam-count returns a count of the expected input parameters for a specified method.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

integer
Returns the number of input parameters that the method requires.

Throws

None.

cntea-get-inparam-name

Syntax

```
(cntea-get-inparam-name hmethod dwIndex)
```

Description

cntea-get-inparam-name queries the input parameter name.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle to the method returned by **cntea-create-method**.
dwIndex	integer	An integer which indexes the input parameter (list starts at 0). See item 2 below for usage of dwIndex.

Return Values

string
Returns the string specifying the next input parameter name.

Throws

None.

Additional Information

1. Use **cntea-get-inparam-count** to obtain a count of the input parameters.
2. Loop through the count calling **cntea-get-next-inparam** to iterate through each parameter.
3. Call this function on each of the iterations to obtain the name of the parameter.

cntea-get-inparam-type

Syntax

```
(cntea-get-inparam-type hmethod dwIndex)
```

Description

cntea-get-inparam-type queries the type of a specified input parameter.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.
dwIndex	integer	An integer which indexes the input parameter (The list starts at 0).

Return Values

integer

Returns an integer indicating the parameter type. Table 2 indicates the type mappings.

Table 2 Type Mappings for Input Parameters

Type	Description
1	FLOAT
2	INTEGER
3	LONG
5	STRING
11	UNKNOWN

Throws

None.

Additional Information

1. Call **cntea-get-inparam-count** to get a count of the input parameters.
2. Loop through the count calling **cntea-get-next-inparam** to iterate through each parameter.
3. Call this function on each of the iterations to obtain the name of the parameter.

cntea-get-outparam-type

Syntax

```
(cntea-get-outparam-type hmethod)
```

Description

cntea-get-outparam-type queries the type of a specified output parameter.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

integer
Returns an integer indicating the parameter type. Table 3 indicates the type mappings.

Table 3 Type Mappings for Output Parameters

Type	Description
1	FLOAT
2	INTEGER
3	LONG
5	STRING
6	TEMPLATE (one or more rows)
10	NO OUTPUT

Throws

None.

cntea-get-outparam-name

Syntax

```
(cntea-get-outparam-name hmethod)
```

Description

cntea-get-outparam-name queries the output parameter name.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

string
Returns a string indicating the name of the output parameter.

Throws

None.

cntea-get-outparam-size

Syntax

```
(cntea-get-outparam-size hmethod)
```

Description

cntea-get-outparam-size queries the size of the specified output parameter.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

integer
Returns the size of the parameter.

Throws

None.

Additional Information

In the case of a "basic" returned type, this returns the size of the "basic" type (i.e., length of a string returned). For an output of type template, this API will return the count of the members of the template.

cntea-get-outparam-member-type

Syntax

```
(cntea-get-outparam-member-type hmethod dwIndex)
```

Description

cntea-get-outparam-member-type queries the type of a template member that the method returns.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Name	Type	Description
dwIndex	integer	An integer which indexes a member of a template (The list of members starts at 0).

Return Values

integer
Returns an integer indicating the parameter type. Table 3 indicates the type mappings.

Table 4 Type Mappings for Output Parameters

Type	Description
1	FLOAT
2	INTEGER
3	LONG
5	STRING
6	TEMPLATE (one or more rows)
10	NO OUTPUT

Throws

None.

Additional Information

1. Call **cntea-get-outparam-type** to determine if returned type is a template.
2. Call **cntea-gat-outparam-size** to determine the number of members.
3. Loop through the count of members, and call this API, passing the current index value through each iteration of the loop to obtain each member's type.

cntea-get-outparam-member-name

Syntax

```
(cntea-get-outparam-member-name hmethod dwIndex)
```

Description

cntea-get-outparam-member-name queries the name of the output of a template member that the method returns.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Name	Type	Description
dwIndex	integer	An integer which indexes a member of a template (The list of members starts at 0).

Return Values

string
Returns a string indicating the name of the output parameter.

Throws

None.

Additional Information

1. Call **cntea-get-outparam-type** to determine if returned type is a template.
2. Call **cntea-get-outparam-size** to determine the number of members.
3. Loop through the count of members calling this API, passing the current index value through each iteration of the loop to obtain each member's name.

cntea-get-outparam-member-size

Syntax

```
(cntea-get-outparam-member-size hmethod dwIndex)
```

Description

cntea-get-outparam-member-size queries the size of a template member that the method returns.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.
dwIndex	integer	An integer which indexes a member of a template (The list of members starts at 0).

Return Values

integer
Returns the size of a template member.

Throws

None.

Additional Information

1. Call **cntea-get-outparam-type** to determine if returned type is a template.

2 Call **cntea-get-outparam-size** to determine the number of members.

3 Loop through the count of members, and call this API, passing the current index value through each iteration of the loop to obtain each member's size.

cntea-set-inparam-value

Syntax

```
(cntea-set-inparam-value hmethod pszInParamName pszValue)
```

Description

cntea-set-inparam-value sets a value for a specified input parameter of a method.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.
pszInParamName	string	The name of the parameter.
pszValue	string	The value for the input parameter.

Return Values

Boolean
Returns **#t** (true) indicating the parameter value set successfully; otherwise, returns **#f** (false).

Throws

None.

Note: *If a particular parameter is optional, you do not have to call this function (cntea-set-inparam-value) to populate the optional parameter. It is incorrect to call this function and populate the parameter with a dummy value such as "" (double quotes).*

cntea-clear-inparam-values

Syntax

```
(cntea-clear-inparam-values hmethod)
```

Description

cntea-clear-inparam-values clears the input parameter list.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

Boolean

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

Throws

None.

cntea-invoke

Syntax

```
(cntea-invoke hmethod secTimeout)
```

Description

cntea-invoke invokes a method.

Parameters

Name	Type	Description
hmethod	opaque method	The handle returned by **cntea-create-method**.
secTimeout	integer	The timeout period expressed in seconds.

Return Values

Boolean

Returns **#t** (true) indicating the method invocation succeeded; otherwise, returns **#f** (false).

Throws

None.

cntea-get-result-name

Syntax

```
(cntea-get-result-name hmethod)
```

Description

cntea-get-result-name queries the name of the output from the result (a "basic" result such as a string).

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

string
Returns the name of the output result.

Throws

None.

Additional Information

Call **cntea-invoke** before calling this API.

The result will either be of the type "basic" or "template."

cntea-get-result-value

Syntax

```
(cntea-get-result-value hmethod)
```

Description

cntea-get-result-value queries the value of the output from the result (a "basic" result such as a string).

Parameters

Name	Type	Description
hmethod	string	The handle returned by **cntea-create-method**.

Return Values

string
Returns the string value of the output.

Throws

None.

Additional Information

Call cntea-invoke before calling this API.

The result will either be of the type "basic" or "template."

cntea-get-result-member-name

Syntax

```
(cntea-get-result-member-name hmethod dwIndex)
```

Description

cntea-get-result-member-name queries the name of a template member from the output.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.
dwIndex	integer	An integer which indexes a member of a template (The list of members start at 0).

Return Values

string
Returns the name of the member.

Throws

None.

Additional Information

1. Call **cntea-invoke**.
2. Ensure that the result returned is a data template type.

cntea-get-result-member-value

Syntax

```
(cntea-get-result-member-value hmethod dwIndex)
```

Description

cntea-get-result-member-value queries the value of a template member from the output.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Name	Type	Description
dwIndex	integer	An integer which indexes a member of a template (The list starts at 0).

Return Values

string
Returns the value of the output.

Throws

None.

Additional Information

1. Call cntea-invoke.
2. Ensure that the result returned is a data template type.

cntea-get-result-row-count

Syntax

```
(cntea-get-result-row-count hmethod)
```

Description

cntea-get-result-row-count queries the count of result rows returned for the result table template.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Return Values

integer
Returns the number of rows.

Throws

None.

Additional Information

1. Call **cntea-invoke**.
2. Use **cntea-get-outparam-type** to determine if the output is of table template type.

cntea-get-result-row

Syntax

```
(cntea-get-result-row hmethod dwIndex)
```

Description

cntea-get-result-row iterates through rows (data template) of a table template making the various rows available for retrieval by calling an additional function, such as **cntea-get-result-row-member** or **cntea-get-result-row-string**.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by cntea-create-method.
dwIndex	integer	An integer which indexes the row in the table template (The list starts at 0).

Return Values

Boolean

Returns **#t** (true) indicating success; otherwise, returns **#f** (false).

Throws

None.

Additional Information

Note: This API sets the pointer to the result row at the position indicated by dwIndex. Use the other methods to get to the member data of the row.

1. Call **cntea-invoke**.
2. Use **cntea-outparam-type** to determine if output is of the table template.
3. Use **cntea-get-outparam-size** to determine the number of members in the template.
4. Use **cntea-get-count-result-row** to determine the number of rows of results.
5. Loop through each row calling this API, passing in the index.

   Within each iteration of a row, loop through the count of members and pass in the member index to the call of **cntea-get-next-result-member** and/or **cntea-get-result-member-value** to get the name and/or the value of each member.

cntea-get-row-result-string

Syntax

```
(cntea-get-row-result-string hmethod pszDL1 pszDL2)
```

Description

cntea-get-row-result-string gets a row of results in string format.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.
pszDL1	string	The parameter used to set the delimiter name and value.
pszDL2	string	The parameter used to set the delimiter name/value pairs.

Return Values

string
Returns a string as the result in the form described below:

```
Name<DL1>Value<DL2>...<DL2>NameN<DL1>ValueN<DL2>
```

Throws

None.

Additional Information

1. Call **cntea-invoke**.
2. Use **cntea-outparam-type** to determine if output is of the data template type.
3. If yes, call this API to get all the members in "one group" in the form of a string.
4. Use **cntea-outparam-type** to determine if output is of the table template type.
5. If yes, call **cntea-get-next-result-row-count** to get the count of rows.
6. Loop through count calling **cntea-get-next-result-row**, for each iteration call this API to get all the members of the row in "one group" in the form of a string.

cntea-get-allrows-result-string

Syntax

```
(cntea-get-allrows-result-string hmethod pszDL1 pszDL2 pszDL3)
```

Description

cntea-get-row-result-string gets all rows of a table result as a string.

Parameters

Name	Type	Description
hmethod	opaque handle	The handle returned by **cntea-create-method**.

Name	Type	Description
pszDL1	string	The parameter used to set the delimiter name and value.
pszDL2	string	The parameter used to set the delimiter name/value pairs.
pszDL3	string	The parameter used to set the row delimiter.

Return Values

string

Returns a string as the result in the form described below:

```
Name<DL1>Value<DL2>...<DL2>NameN<DL1>ValueN<DL2><DL3>
```

Throws

None.

Additional Information

1. Call **cntea-invoke**.
2. Call **cntea-outparam-type** to determine if the output is of the template type (result is one or more rows), if it is, then
3. Call this function to get all the rows returned as a string in "one group."

Index

N

P

R

S

U

Z