SeeBeyond™ eBusiness Integration Suite

e*Way Intelligent Adapter for Clarify User's Guide

Release 4.5.3

Version 20021030142009.

Contents

Preface

This Preface contains information regarding the user's guide.

0.1 Intended Reader

The reader of this user's guide is presumed:

- To be a developer or system administrator with responsibility for maintaining the SeeBeyond™ e*Gate™ Integrator system
- To be familiar with Windows-style GUI operations
- To be familiar with Clarify eFrontOffice
- To have knowledge of Windows NT, Windows 2000, or UNIX operations and administration

0.2 Organization

This user's guide is organized roughly into two parts. The first part, consisting of Chapters 1 through 4, introduces the e*Way™ intelligent adapter and describes the procedures for installing the e*Way and implementing a working system incorporating the e*Way.

Chapter 4 also includes a description of a sample schema, which can be used to test your system following installation and, if appropriate, as a template that you can modify to produce your own schema. This part should be of particular interest to a system administrator or other user charged with the task of getting the system up and running.

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

0.3 Nomenclature

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

0.4 Online Viewing

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

Hyperlinking is available in:

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

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

Chapter 1

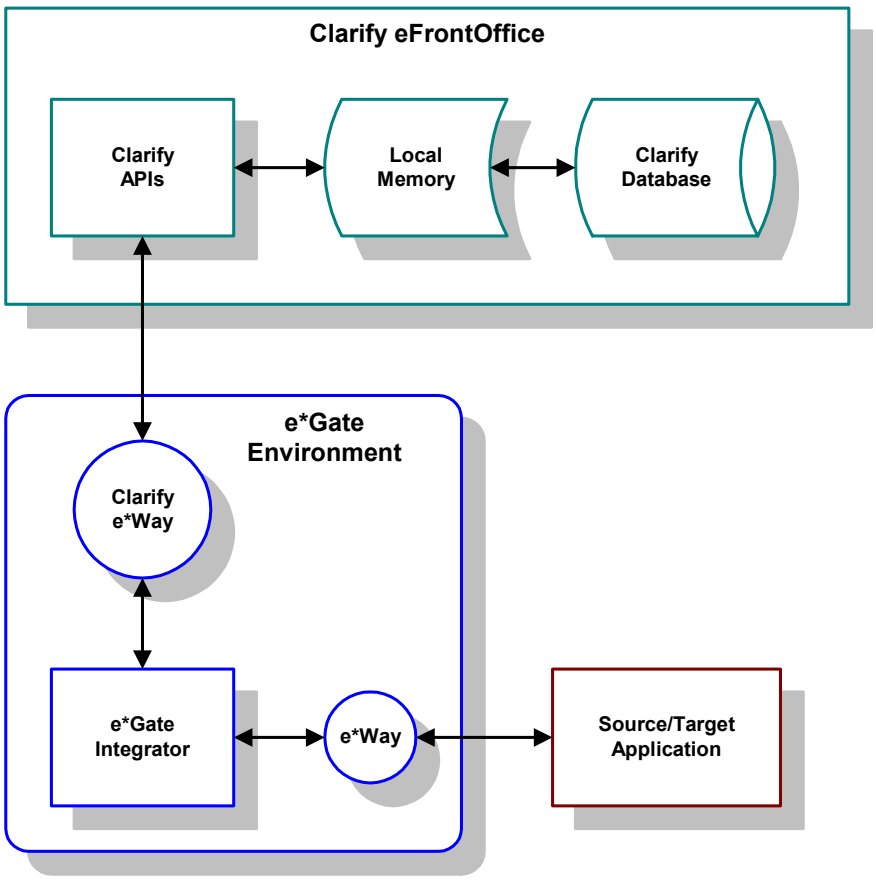
Introduction

The e*Way Intelligent Adapter for Clarify enables e*Gate Integrator to exchange data with Clarify eFrontOffice applications.

1.1 Overview

Figure 1 diagrams the data flow between Clarify eFrontOffice and an external source or target application, through e*Gate.

Figure 1 Clarify e*Way Data Flow

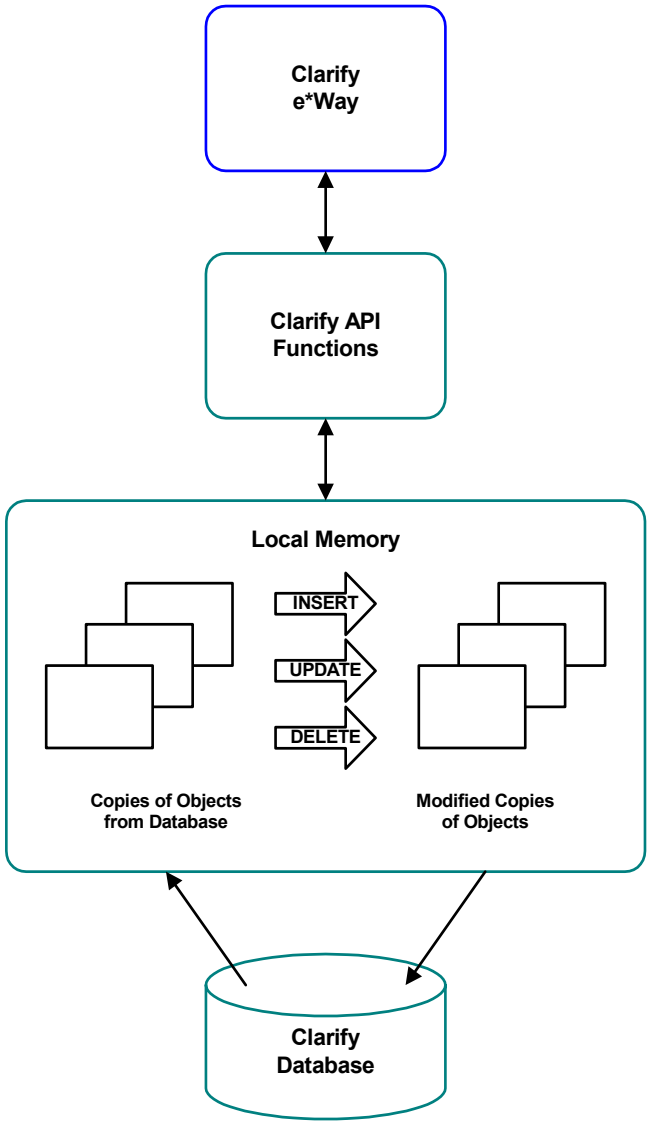


1.2 The Clarify e*Way Interface

The Clarify e*Way interacts with a Clarify server through the Clarify Application Programming Interface (API). The Clarify API consists of a set of function calls made available by Clarify, which work in concert with local memory to process transactions to and from the Clarify database.

The Clarify API creates a space in local memory called a *transaction state*. Data is transferred from the database to this space, where the Clarify application modifies the information. When these changes are completed, the Clarify application commits the transaction state to the database and saves the changes made in the objects to the database.

Figure 2 Clarify Interface



The Clarify API manages the local memory, and allows the Clarify e*Way to pass data to and from local memory for transfer to the Clarify database. The Insert, Update, and Delete functions perform all necessary data transactions. The functionality of the Clarify e*Way is shown in **Figure 2 on page 12**.

For information regarding the Clarify 10.1 HLAPI or high-level API's and low-level API's, see "High-Level API (HLAPI) Programmer's Guide" and the "API Toolkit Technical Reference" in your Clarify eFrontOffice 10.1 documentation.

1.3 Components

The Clarify e*Way includes the following components:

- An executable file (Generic e*Way Kernel), **stcewgenericmonk.exe**
- For Clarify 8, an accompanying dynamic load library, **stc_monkclarify.dll**, which extends the Generic e*Way Kernel to form the Clarify e*Way
- For Clarify 10.1, an accompanying dynamic load library, **stc_monkclarify_10.dll,** which extends the Generic e*Way Kernel to form the Clarify e*Way; for the UNIX operating system on Clarify 10.1, a static load library, **stcewclarifymonk.exe** is used to extend the Generic e*Way Kernel to form the Clarify e*Way
- A default configuration file, **stcewclarify.def**
- Monk function scripts and library files, discussed in **Chapter 7**
- Sample schema, discussed in **"Sample Schema" on page 32**

A complete list of installed files appears in **Table 1 on page 17**.

1.4 Supported Platforms

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

- Windows 2000, Windows 2000 SP1, Windows 2000 SP2, and Windows 2000 SP3
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8

1.5 System Requirements

To use the Clarify e*Way, you need the following requirements:

- An e*Gate Participating Host, version 4.5.1 or later.
- A TCP/IP network connection.
- The database client for your specific database underlying Clarify.

The client components of the databases with which the e*Way interfaces have their own requirements; see that system's documentation for more details.

1.6 External System Requirements

Clarify eFrontOffice, version 8.0 or 10.1.

The Clarify e*Way and the Clarify command line ETD Builder require the appropriate database client installation on the system where the e*Way and the Builder is executed from.

Refer to the Clarify documentation for specific underlying database platform version information. For example, if your underlying database is Oracle, Clarify 10.1 requires Oracle 8i.

Chapter 2

Installation

This chapter describes the requirements and procedures for installing the e*Way Intelligent Adapter for Clarify.

2.1 Windows NT or Windows 2000

Caution: ***Do not*** *edit any installation scripts or change the suggested "installation directory" setting without instructions from SeeBeyond.*

2.1.1 Installation Procedure

1. Log in as an Administrator on the workstation on which you want to install the e*Way.

Note: *You must have Administrator privileges to install this e*Way.*

2. Exit all Windows programs and disable any anti-virus applications before running the setup program.
3. Insert the e*Way installation CD-ROM into the CD-ROM drive.
4. If the CD-ROM drive's **Autorun** feature is enabled, the setup application should launch automatically. If not, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the installation CD-ROM.
5. The InstallShield setup application launches. Follow the on-screen instructions to install the e*Way.

Note: *The installation utility detects and suggests the appropriate installation directory. Use this directory unless advised otherwise by SeeBeyond.*

Once you have installed this e*Way, you must configure it for your system and incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types.

2.1.2 Environment Configuration

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

2.1.3 External Configuration Requirements

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

2.2 UNIX

2.2.1 Installation Procedure

You must select both the Windows NT or Windows 2000 and the UNIX operating systems during the installation process. This allows you to use the command line ETD Builder which requires Windows NT or Windows 2000.

Note: *You are not required to have 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.*

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

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type

   cd /cdrom
4. Start the installation script by typing:

   setup.sh
5. A menu of options appears. 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.***

2.2.2 Environment Configuration

On UNIX platforms, assure that your **LD_LIBRARY_PATH** includes the appropriate database client library location for your Clarify application.

For example:

```
/db/<oracle8>/lib
```

Where:

<oracle8> equals your database.

Any modification must be made prior to any attempt to run the Clarify e*Way.

2.2.3 External Configuration Requirements

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

2.3 Subdirectories and Files

The Clarify e*Way installation process creates the subdirectories and installs the files shown in Table 1 within the **eGate\client** tree on the Participating Host. The files are then committed to the "default" schema on the Registry Host.

Table 1 Subdirectories and Files

Subdirectories	Files
For Clarify 8 bin\	stc_monkclarify.dll libdb_b.dll mny204d.dll stcclarifconvert.exe tls704d.dll stcewgenericmonk.exe stc_dbora8_clf.dll stc_dbora8i_clf.dll
For Clarify 10.1 Windows bin\	stc_monkclarify_10.dll libdb_b.dll stcclarify10convert.exe stc_dbora8_clf.dll stc_dbora8i_clf.dll stcewclarifymonk.exe
For Clarify 10.1 UNIX bin\	 stcewclarifymonk.exe

Table 1 Subdirectories and Files (Continued)

Subdirectories	Files
configs\stcewclarify\	stcewclarify.def
monk_library\	clarify-clear-error.monk clarify-complex-query.monk clarify-error-process.monk clarify-finish.monk clarify-get-error.monk clarify-relation-query.monk clarify-simple-query.monk clarify-struct-insert.monk clarify-struct-lookup.monk clarify-struct-update.monk clarify.monk clarify-int.monk clarify-init_10.monk clarify-init_10_sol.monk

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

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

2.4 Optional Example Files

2.4.1 Installation Procedure

The installation CD contains a sample schema in the **samples\ewclarify** directory, which must be loaded on your system manually if you wish to use it. The sample schema is discussed in detail in **Sample Schema** on page 32.

- **ClarifyPartNumPost**: Schema example for Products data.

2.4.2 Subdirectories and Files

To load the sample schema ClarifyPartNumPost.zip:

1. Start e*Gate Enterprise Manager.
2. Create a new schema.
3. On the **File** menu, click **Import Definitions from File...**

4 Click **Next**.

5 Click the **Schema** option button.

6 Click the **Select** button

7 Navigate to the CD-ROM drive where the e*Gate installation disk is located.

8 In the **Sample** folder, select **ClarifyPartNumPost.zip** and click **Open**.

9 Click **Next**.

10 Click **Finish**.

The previous procedure creates the subdirectories and installs the files shown in Table 2.

Table 2 Created Subdirectories and Files

Subdirectories	Files
\<clarify_sample>	Clarify-Part-Num.cfg Clarify-Part-Num.sc ClarifyPartNumPost.ctl ClarifyPartNumPost.exp Clarify-Part-Num-Post.tsc Sample-Feeder.cfg Sample-Feeder.sc Sample-Part-Num.ssc table_part_num.ssc

Chapter 3

e*Way Setup

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

3.1 Overview

After installing the Clarify e*Way, you must perform an initial setup for it to work with your system. A wide range of setup options allow the e*Way to conform to your system's operational characteristics and your facility's **operating** procedures.

The topics discussed in this chapter are:

Using the e*Gate Enterprise Manager

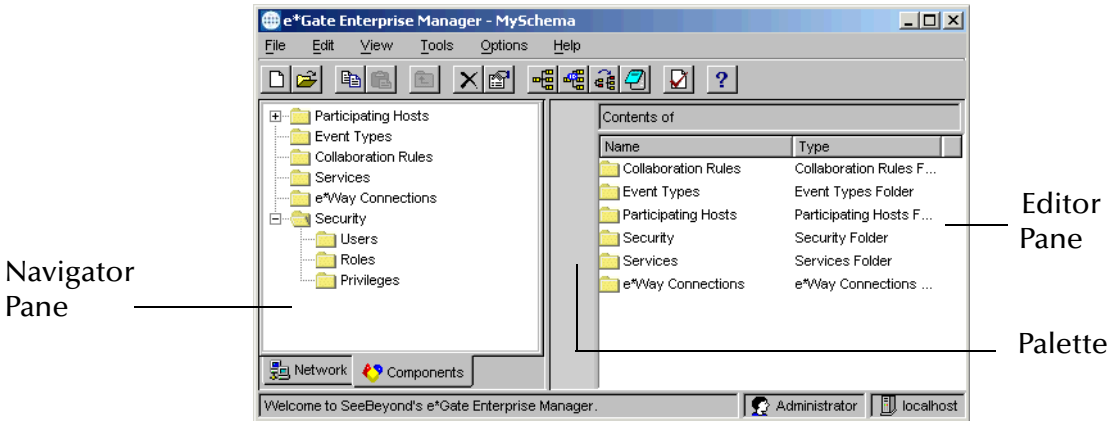
Setting Up the e*Way

Troubleshooting the e*Way

3.2 Using the e*Gate Enterprise Manager

First, here is a brief introduction to the e*Gate Enterprise Manager. The general features of the e*Gate Enterprise Manager window are shown in Figure 3.

Figure 3 e*Gate Enterprise Manager Window (Components View)



Use the Navigator and Editor panes to view the e*Gate components. You can only view the components of a single schema at one time, and all operations apply only to the current schema. Specialized command buttons (see Table 3) appear in the Palette area of the window, depending on which levels of the Components Tree are open. For additional information, see the *e*Gate Integrator User's Guide*.

Table 3 Setup Command Buttons

Button	Name	Location	Function
	**Create e*Way**	Palette	Creates a new e*Way
	**Create IQ**	Palette	Creates a new Intelligent Queue (IQ)
	**Create BOB**	Palette	Creates a new Business Object Broker
	**Create e*Insight Engine**	Palette	Creates a new e*Insight Engine (only if e*Insight BPM is installed)

3.3 Setting Up the e*Way

3.3.1 Defining e*Way Components

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

To create an e*Way

1. Select the e*Gate Enterprise Manager Navigator's **Components** tab.
2. Open the host on which you want to create the e*Way.
3. Select the Control Broker that manages the new e*Way.
4. On the **Palette**, click **Create**.
5. Enter the name of the new e*Way, then click **OK**.

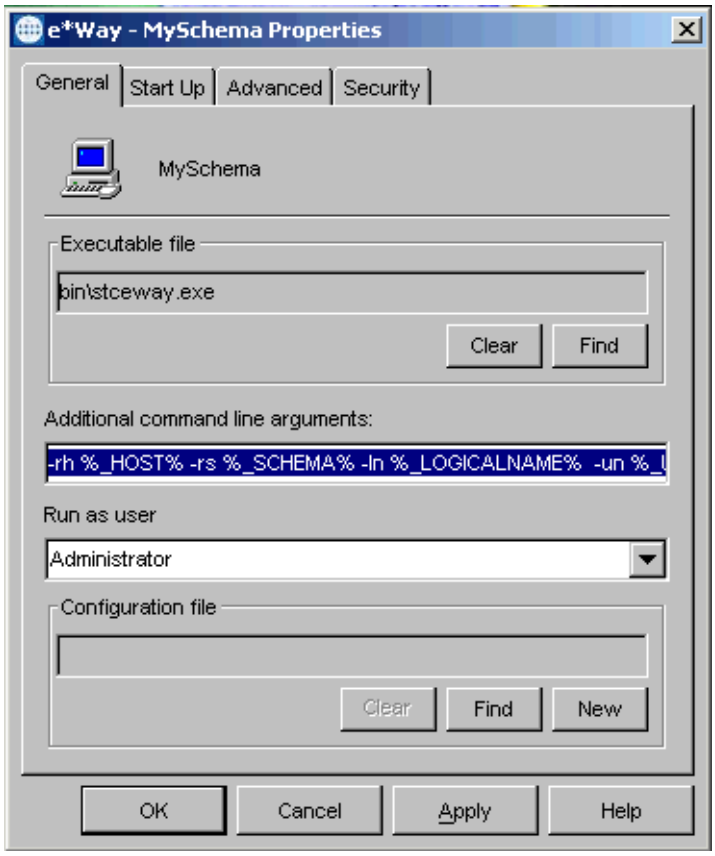
3.3.2 Modifying e*Way Properties

Note: *Selecting the executable file should be the first configuration procedure you perform once you have created the e*Way.*

To modify any e*Way properties

1. Select the Navigator's **Components** tab.
2. Open the host on which the desired e*Way runs.
3. Open the Control Broker that manages the e*Way.
4. Select the desired e*Way.
5. Right-click on the e*Way and select **Properties** to edit the e*Way's properties. The properties dialog box opene to the **General** tab (shown in **Figure 4 on page 23**).

Figure 4 e*Way Properties (General Tab)



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

Note: *When you shut down an e*Way and open its property sheet in the e*Gate Enterprise Manager, once you click **OK** or **Apply**, the e*Way immediately restarts. This action only happens if the e*Way is in autostart mode. After you click **OK** or **Apply**, the Registry is automatically updated with any changes, if you made them using the e*Way Editor.*

3.3.3 Selecting an Executable File

Selecting the executable file is the first and most important step in configuring the e*Way. This step determines what type of e*Way runs and thus what type of external system or communications protocol it supports.

You must know which executable file to select before you perform this procedure.

To select an e*Way's executable file

1 Display the e*Way's properties (see the **procedure on page 22**).

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

3 Use the file selection dialog box to select the executable files. All e*Way executable files have an **.exe** extension.

Note: *You must use the **Find** button to select the executable file. You cannot type its name directly into the Executable File box.*

3.3.4 Selecting or Creating a Configuration File

After you have selected an executable file, you must select or create a configuration file that contains the operating parameters for the e*Way.

To select an existing configuration file

1. Display the e*Way's properties (see the **procedure on page 22**).
2. On the **General** tab, under **Configuration File**, click **Find**.
3. Use the file selection dialog box to select the desired file (**.cfg**).
4. Exit the e*Way Editor.

Note: *You must use the **Find** button to select the configuration file. You cannot type its name directly into the Configuration File box.*

To create a configuration file

1. Display the e*Way's properties (see the **procedure on page 22**).
2. On the **General** tab, under **Configuration File**, click **Find**.
3. Use the file selection dialog box to select a default configuration (template) file.

Note: *All e*Way default configuration files have a **.def** extension, and are intended to be used as templates.*

4. Use the e*Way Editor to change the default configuration parameters as required (see **Chapter 6**).
5. Edit the **Additional Command Line Arguments** box to include any arguments you require (see the following procedure).
6. Save the file with a **.cfg** extension and exit the e*Way Editor.

Note: *You must use the **Find** button to select the configuration file. You cannot type its name directly into the Configuration File box.*

3.3.5 Changing Command-line Parameters

Most SeeBeyond e*Ways require only the default command-line parameters shipped with the e*Gate Enterprise Manager. Use the procedure in this section only if the e*Way you are configuring requires special command-line options, or if you are directed to do so by SeeBeyond support personnel.

To change an e*Way's command-line options

1. Display the e*Way's properties (see the **procedure on page 22**).

2 On the **General** tab, edit the **Additional Command Line Arguments** box to include the arguments you require. Unless you have a specific need to do so, do not change any of the existing parameters; append any new parameters to the end of the command line.

3.3.6 Changing the User Name

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

To change the user name

1 Display the e*Way's properties (see the **procedure on page 22**).

2 On the **General** tab, use the **Run As** list to select the e*Gate user under whose name this component runs.

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

3.3.7 Setting Startup Options or Schedules

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

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

You determine how the Control Broker starts or shuts down an e*Way using options on the e*Way properties **Start Up** tab (see **Figure 5 on page 26**). See the *e*Gate Integrator System Administration and Operations Guide* for more information about how interactive monitors can start or shut down components.

Figure 5 e*Way Properties (Start-Up Tab)

To determine whether the e*Way starts automatically when the Control Broker starts

1. Display the e*Way's properties (see the **procedure on page 22**).
2. Select the **Start Up** tab.
3. To activate this feature, check **Start automatically**.
4. To deactivate this feature, clear the **Start automatically** check box.
5. Click **OK**.

To determine whether the e*Way is restarted automatically

1. Display the e*Way's properties (see the **procedure on page 22**).
2. Select the **Start Up** tab.
3. To activate this feature, check **Restart after abnormal termination** and set the desired number of retries and retry interval.
4. To deactivate this feature, clear the **Restart** check box.
5. Click **OK**.

Note: *The **auto restart** feature does not automatically restart the e*Way if the e*Way is shut down manually by an interactive monitor.*

*If the e*Way is shut down and you make any configuration changes using the e*Gate Enterprise Manager, the Control Broker automatically restarts the e*Way when the configuration changes are recorded in the e*Gate Registry. If you do not want the e*Way to restart when configuration changes are made, disable this feature before configuring the e*Way.*

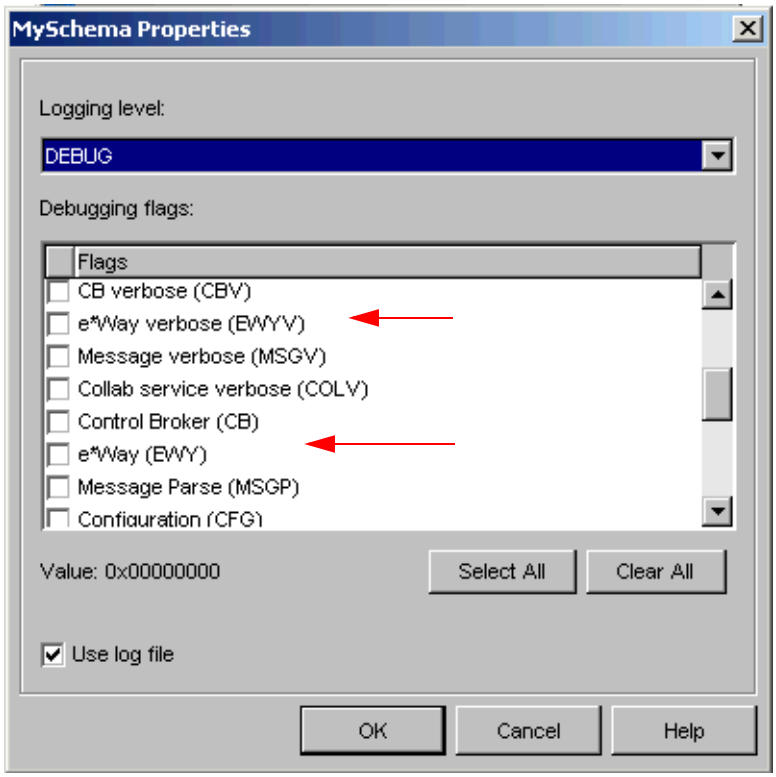
3.3.8 Activating or Modifying Logging Options

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

To set the e*Way debug level and flag

1. Select the Navigator's **Components** tab.
2. Open the host on which the desired e*Way runs.
3. Open the Control Broker that manages the e*Way.
4. Select the desired e*Way.
5. On the Toolbar, click **Properties** to edit the e*Way's properties.
6. Select the **Advanced** tab.
7. Click **Log**. The dialog box appears as shown in **Figure 6 on page 28**.
8. Select **DEBUG** for the **Logging level**.
9. Select either **e*Way (EWY)** or **e*Way Verbose (EWYV)** for the **Debugging flag.** Note that the verbose flag can slow down system performance.
10. Click **OK**.

Figure 6 e*Way Properties (Advanced Tab - Log Option)



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

3.3.9 Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way sends a Monitoring Event to the Control Broker, which is then routed to the e*Gate Monitor or any of a number of destinations.

1. Display the e*Way's properties (see the **procedure on page 22**).
2. Select the **Advanced** tab.
3. Click **Thresholds**.
4. Select the desired threshold options and click **OK**.

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

3.4 Troubleshooting the e*Way

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

3.4.1 Configuration Problems

In the e*Gate Enterprise Manager

Check the following items:

- Does the e*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration Rules employed by this e*Way's Collaborations?
- Do those Collaborations subscribe to and publish Events appropriately?
- Are all the components that "feed" this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e*Way "feeds" properly configured, and are they subscribing to the appropriate Events correctly?

In the e*Way Editor

Check that *all* configuration options are set appropriately. Be sure that any settings that you changed are set correctly; that you have overlooked no required changes; and that any defaults are acceptable for your installation.

On the Participating Host supporting the e*Way

Check that the Participating Host is operating properly, and that it has sufficient disk space to hold the IQ data that this e*Way's Collaborations publish.

In the external application with which the e*Way communicates

Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately. Also check that the connection between the external application and the e*Way is functioning appropriately.

Once the e*Way is up and running properly, operational problems can be due to external influences (network or other connectivity problems), problems in the operating environment (low disk space or system errors), problems or changes in the data the e*Way is processing. There may also be corrections required to Collaboration Rules scripts that become evident in the course of normal operations.

One of the most important tools in the troubleshooter's arsenal is the e*Way log file. See the *e*Gate Integrator System Administration and Operations Guide* for an extensive discussion of log files, debugging options, and how to use the e*Gate monitoring system to monitor both operations and performance.

3.4.2 Password Problems

The e*Way's configuration file contains the correct password, but the remote server reports that the password is incorrect.

The e*Way Editor encrypts the password based upon the entry in the **User Name** configuration field; this problem can occur if the correct password is entered before the user name is entered.

1. In the e*Gate Enterprise Manager, display the e*Way's properties.
2. Under **Configuration file,** click **Edit.**
3. The e*Way Editor launches. Use the parameter- and section-selection controls to navigate to the user-name and password options.
4. Make sure that the correct user name is entered (re-enter it if necessary).
5. Enter the password, replacing any entry that may already exist.
6. Save the configuration file, and exit the e*Way Editor.
7. When you return to the e*Way's property sheet in the e*Gate Enterprise Manager, click **OK** to apply the changes and close the property sheet.
8. Restart the e*Way if necessary.

Chapter 4

System Implementation

In this chapter we take a more detailed look at the information presented in the Introduction, and describe the steps required for setting up a working system. Refer to the SeeBeyond publication *Creating an End-to-End Scenario with e*Gate Integrator* for additional information.

4.1 Overview

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

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

The topics discussed in this chapter include:

4.2 Run-time Initiation Procedure

The following steps describe the procedure for initiating a typical working session using the Clarify e*Way and the Clarify API.

During the first use of the Clarify e*Way, the Clarify database performs an initialization process. The Clarify API's write the schema information to a file called XXXXXXXX.048 where "XXXXXXXX" is the first eight characters in the database name. The amount of time required to create the initialization file can vary.

1. Log into the Clarify database. This step requires that you provide the name of the Clarify database server and database, and a login name/password for the Clarify database.
2. Create a transaction state in local memory to serve as a working space for data accessed from the Clarify database.
3. Find the instance in the Clarify database, or create a new object instance. A copy of the instance is placed in the transaction state in local memory and made available for processing.
4. Modify the copy of the object instance in the transaction state in local memory.
5. Free the results of the query from local memory.
6. Commit the transaction state to the Clarify database.
7. Roll back the transaction state, freeing local memory.
8. Log out of the Clarify database.

4.3 Sample Schema

A sample implementation is available in the **samples\ewclarify** directory of the e*Gate CD-ROM.

- **ClarifyPartNumPost**: Clarify-to-e*Gate example.

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

To use a sample schema

1. Install the sample schema.
2. From the control panel, start the newly registered Control Broker as follows:
   A. Double-click **Services**.
   B. Locate **e*Gate Control Broker <*name of schema*>** and select it.
   C. Click **Start**.

4.4 ETD Builder

The Clarify e*Way provides you with a command-line ETD Builder. Use this tool as follows:

From Windows

1. On the **Start** menu, click **Run**.
2. Type **cmd** and click **OK.**
3. At the command prompt type:

```
c:\eGate\client\bin>stcclarifyconvert.exe <Database Type>
    <Database Server> <Clarify Server> <User> <Password>
    <Clarify Object Name> <Output file>
```

Where the parameters for **stcclearifyconvert.exe** are:

- **Database Type**: Your database type, for example **Oracle8**.
- **Database Server**: The name of your database server.
- **Clarify Server**: The name of your Clarify server.
- **User**: Your user name.
- **Password**: Your password.
- **Clarify Object Name**: The name of your Clarify object, for **example part_num**.
- **Output File**: The name of the output file, for example, **part_num.ssc.**

4.4.1 ClarifyPartNumPost

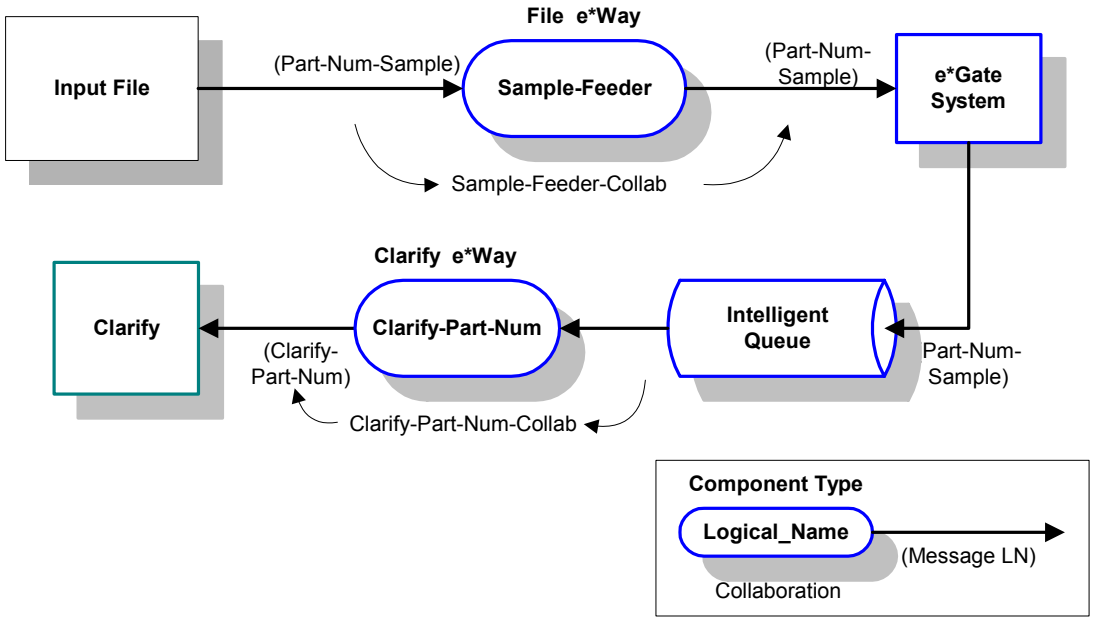
The sample schema **ClarifyPartNumPost** sets up a single instance of the Clarify e*Way and one of the file e*Way, having the logical names shown in Table 4.

Table 4 e*Way Names

e*Way Type	e*Way LN
Clarify e*Way	Clarify-Part-Num
File e*Way	Sample-Feeder

It also sets up an Intelligent Queue (IQ), with the logical name **Clarity-Part-Num**, as shown in **Figure 7 on page 34**.

Figure 7 ClarifyPartNumPost Sample Schema

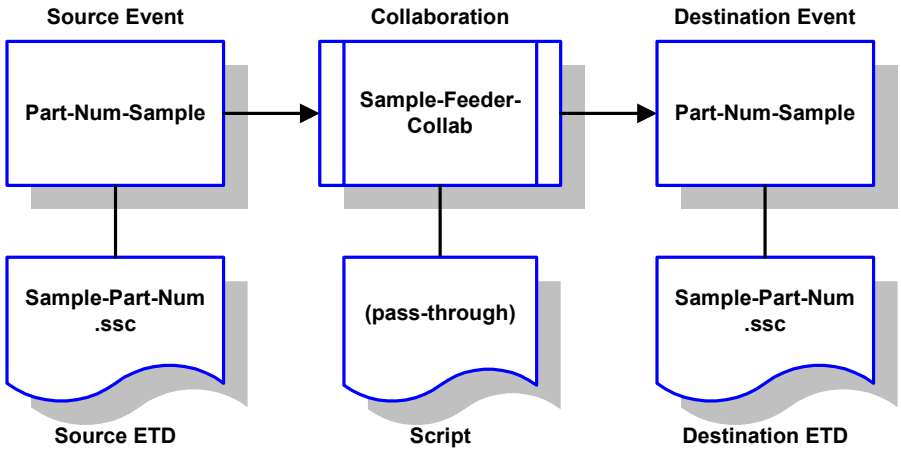


Collaborations

The sample schema contains two Collaborations. The first, **Sample-Feeder-Collab**, is a pass-through Collaboration service that receives a file from Clarify and publishes it to e*Gate (see Figure 8). The second, **Clarify-Part-Num-Collab**, subscribes to the IQ, transforms the data contained in the file and sends it to Clarify (see **Figure 9 on page 35**).

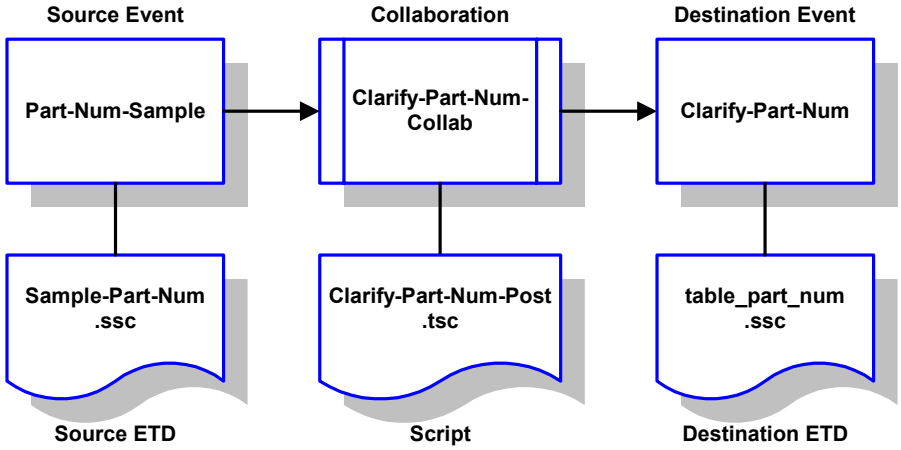
Sample-Feeder-Collab

Figure 8 Sample-Feeder-Collab Collaboration



Clarify-Part-Num-Collab

Figure 9 Clarify-Part-Num-Collab Collaboration



Chapter 5

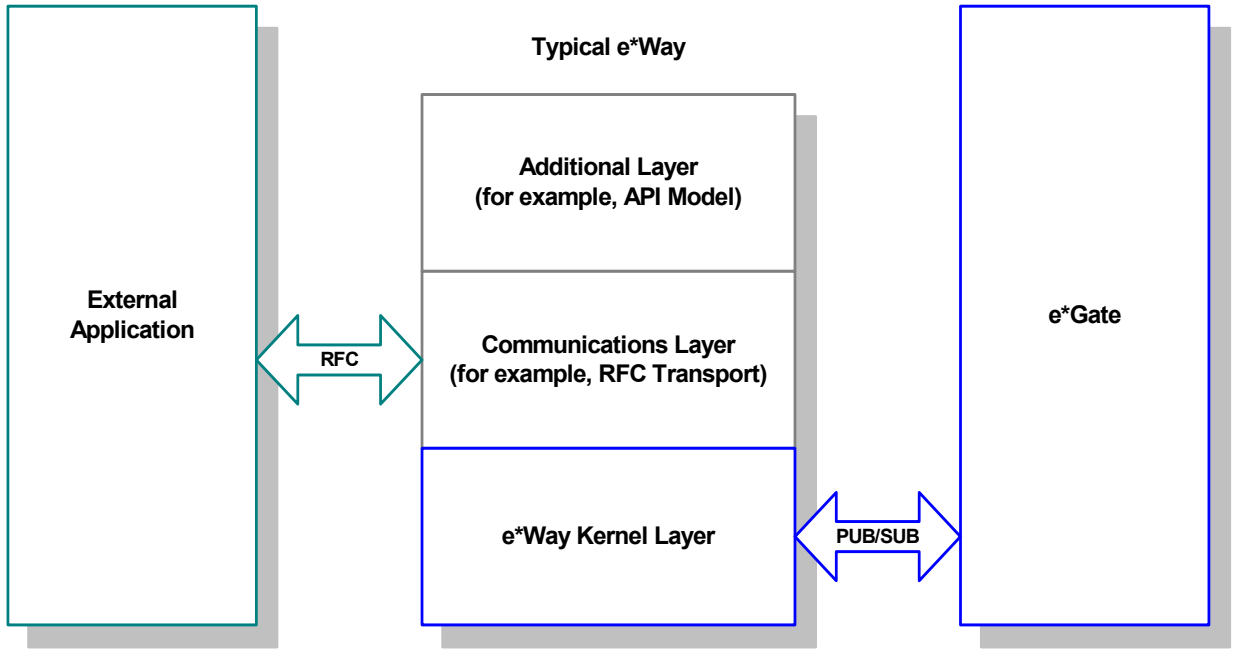
e*Way Operation

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

5.1 Typical e*Way Architecture

Conceptually, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers that handle communication with the external application, built upon an e*Way Kernel layer that manages the processing of data and subscribing or publishing to other e*Gate components (see Figure 10).

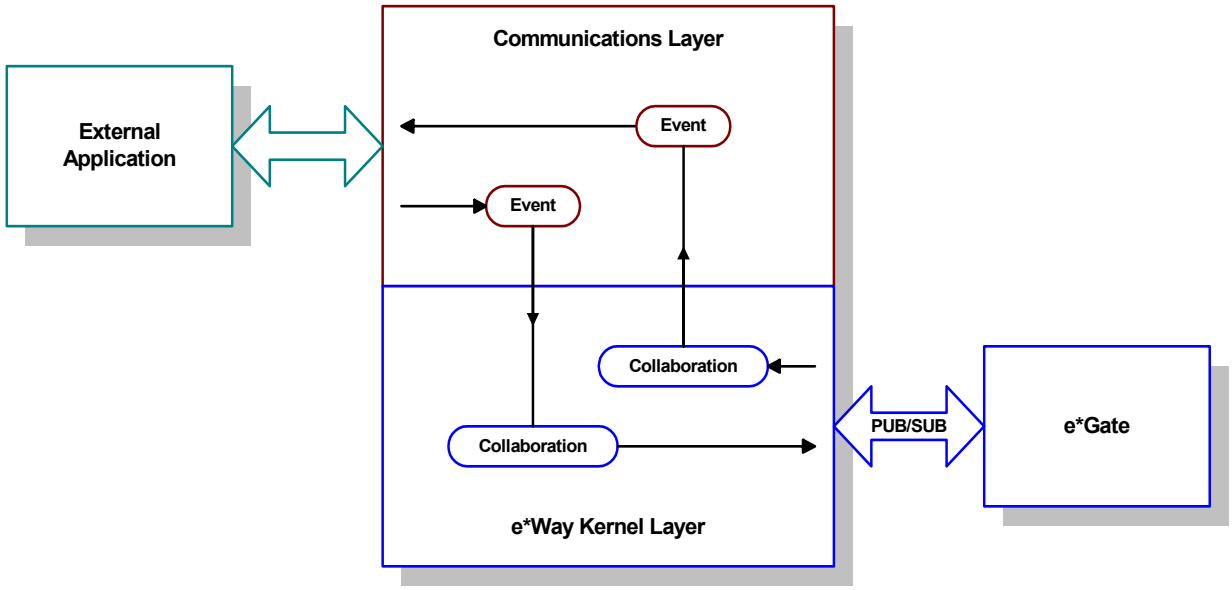
Figure 10 Typical e*Way Architecture



Each layer contains Monk scripts and/or functions, and makes use of lower-level Monk functions residing in the layer beneath. You, as user, primarily use the highest-level functions, which reside in the upper layer(s).

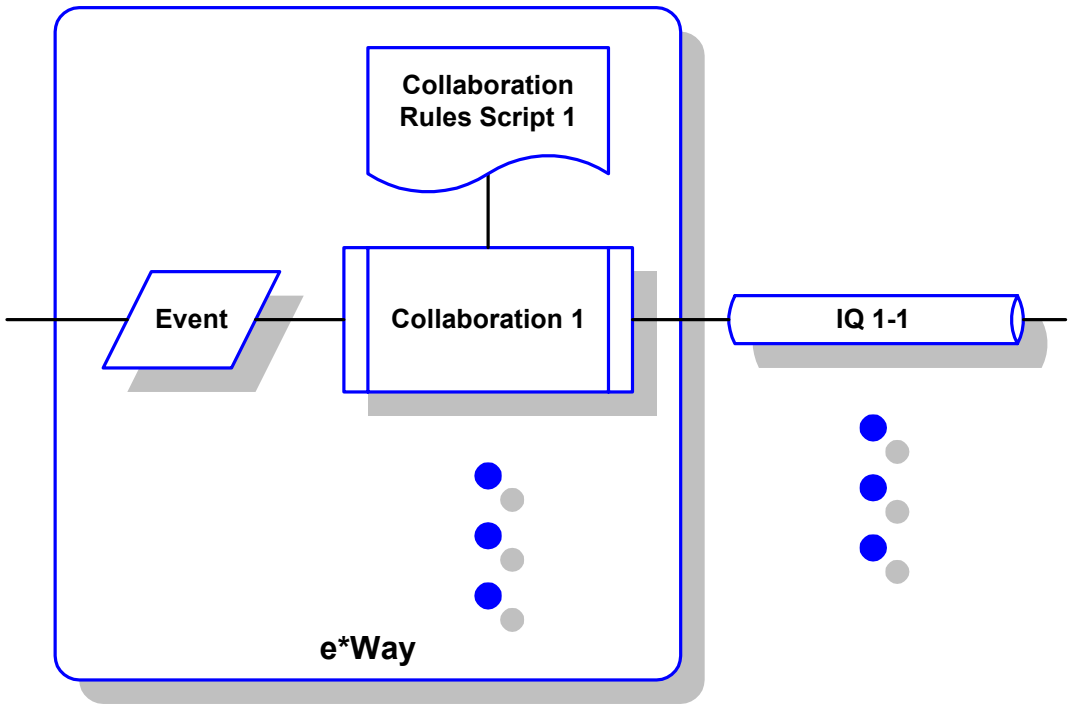
The upper layers of the e*Way use Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events," send those Events to Collaborations, and manage the connection between the e*Way and the external system (see Figure 11).

Figure 11 Basic e*Way Operations



Collaborations execute the business logic that enable the e*Way to do its intended work. In turn, each Collaboration executes a Collaboration Rules script, containing the actual instructions to execute the business logic. Each e*Way requires at least one Collaboration (more, if necessary to perform the required tasks), and each Collaboration requires one or more IQs to which its processed Events are published (see Figure 12).

Figure 12 Collaborations and IQs



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

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

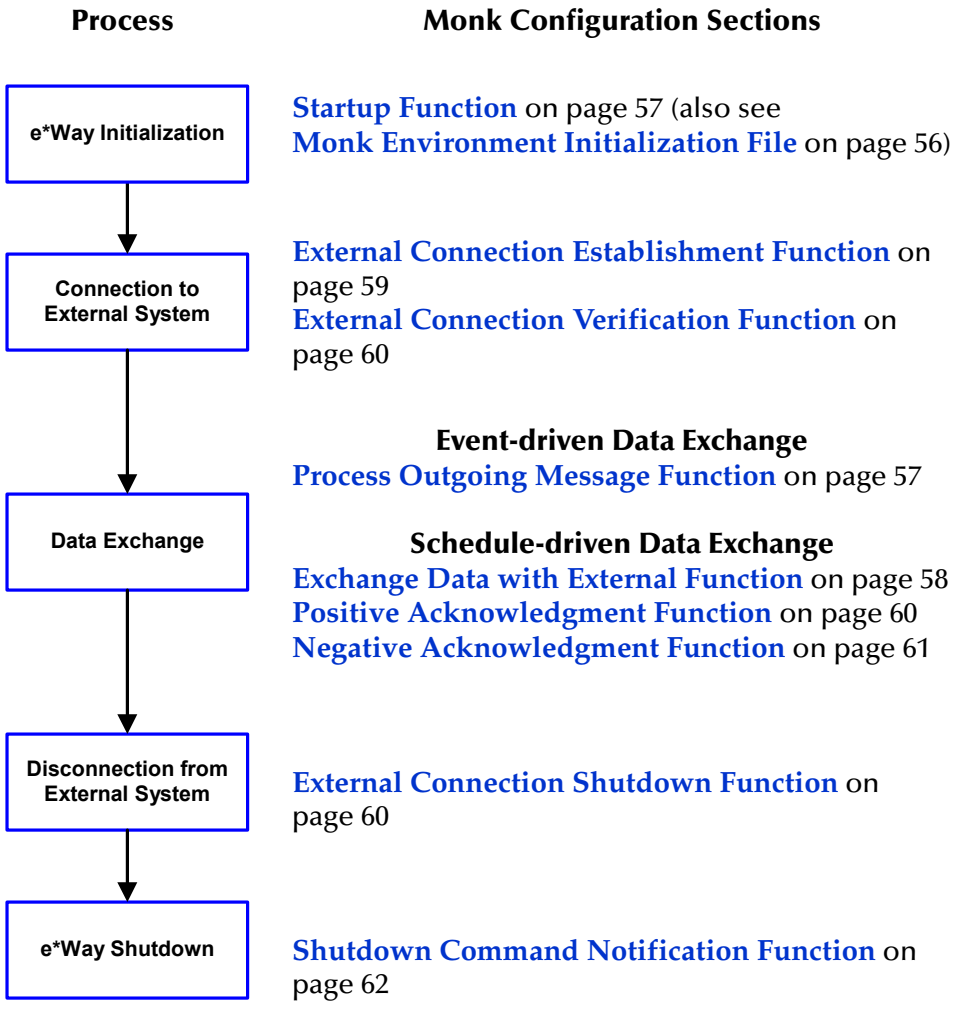
For more information on defining Collaborations, defining IQs, assigning Collaborations to e*Ways, or configuring Collaborations to publish Events, see *Creating an End-to-End Scenario with e*Gate Integrator*.

5.2 Basic e*Way Processes

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

The most basic processes carried out by an e*Way are listed in Figure 13. In e*Ways based on the Generic Monk e*Way Kernel (using **stcewgenericmonk.exe**), these processes are controlled by the listed Monk functions. Configuration of these functions is described in the referenced sections of this User's Guide.

Figure 13 Basic e*Way Processes

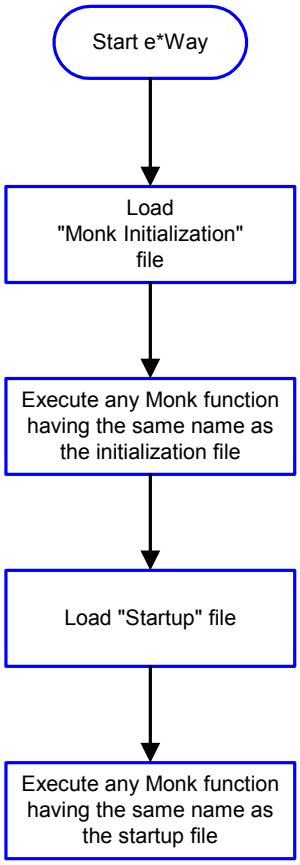


A series of diagrams on the next several pages illustrate the interaction and operation of these functions during the specified processes. Configuring the parameters associated with these functions is covered in **Chapter 6**, while the functions themselves are described in **Chapter 7**.

Initialization Process

Figure 14 illustrates the e*Way's initialization process, using the **Monk Environment Initialization File** and **Startup Function**.

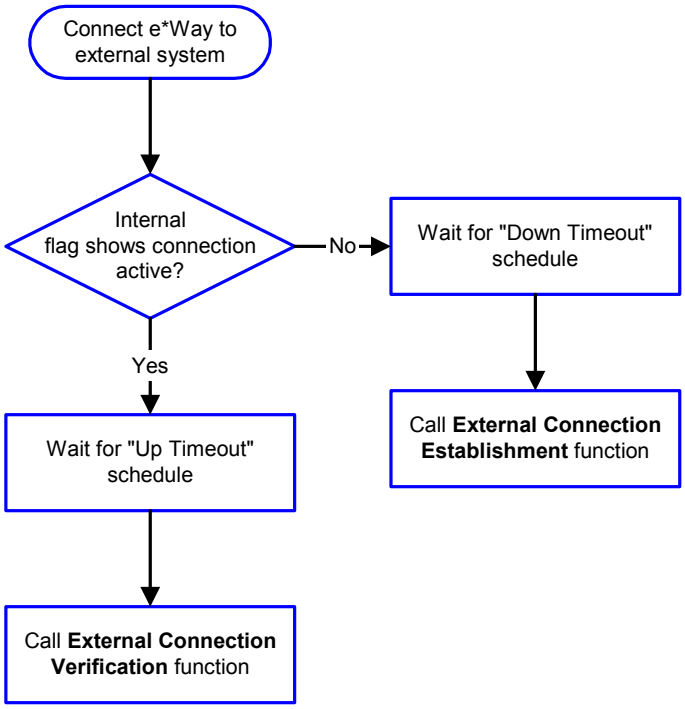
Figure 14 Initialization Process



Connect to External Process

Figure 15 illustrates how the e*Way connects to the external system, using the **External Connection Establishment Function** and **External Connection Verification Function**.

Figure 15 Connection Process



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 16 on page 42** *and* **Figure 17 on page 44** *for examples of how different functions use this flag.*

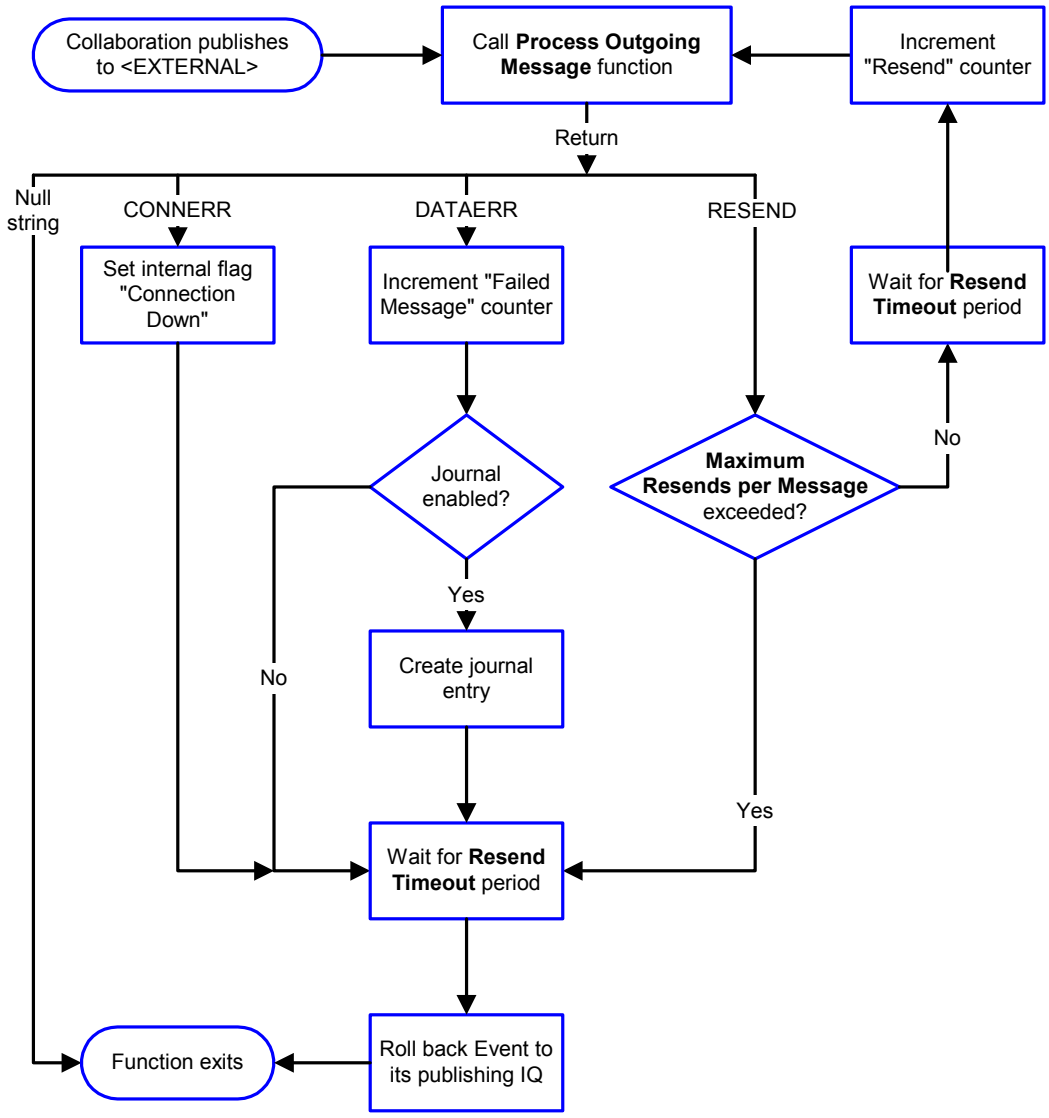
User functions can manually set this flag using Monk functions. See **send-external-up** *on page 196 and* **send-external-down** *on page 197 for more information.*

Data Exchange Process

Event-driven

Figure 16 illustrates how the e*Way's event-driven data exchange process works, using the **Process Outgoing Message Function**.

Figure 16 Event-Driven Data Exchange Process



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 **Process Outgoing Message Function** exits, the e*Way waits for the next outgoing Event.

Schedule-driven

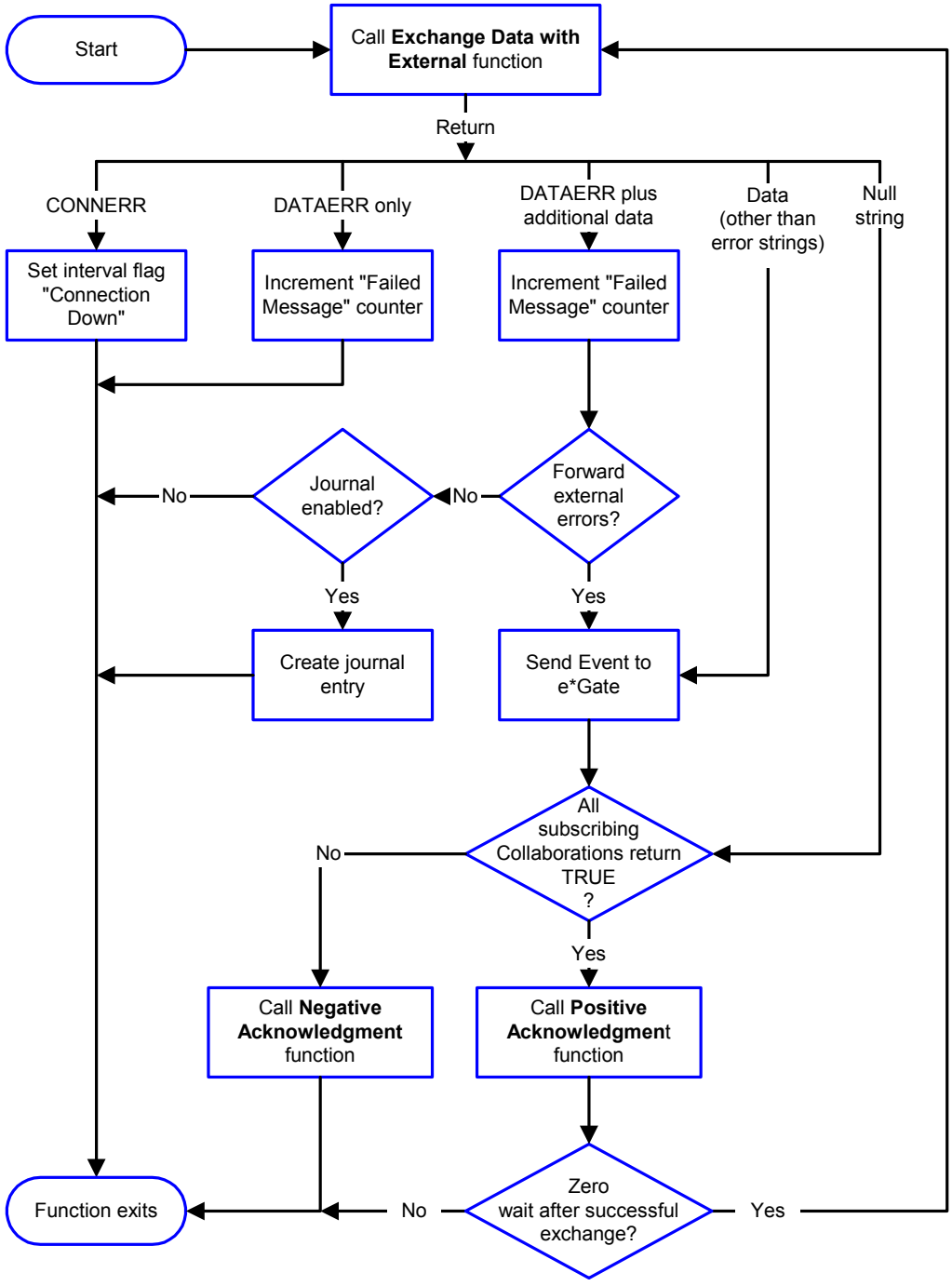
Figure 17 on page 44 illustrates how the e*Way's schedule-driven data exchange process works, using the **Exchange Data with External Function**, **Positive Acknowledgment Function**, and **Negative Acknowledgment Function**.

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

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

After the **Exchange Data with External Function** exits, the e*Way waits for the next "start schedule" time or command.

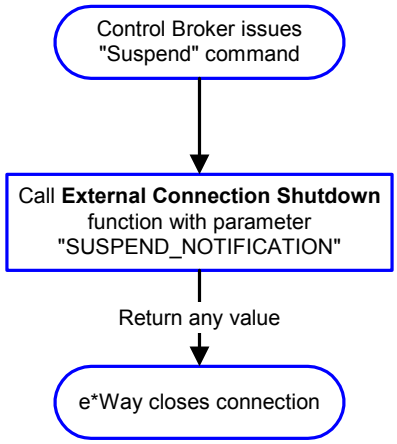
Figure 17 Schedule-Driven Data Exchange Process



Disconnect from External Process

Figure 18 illustrates how the e*Way disconnects from the external system, using the **External Connection Shutdown Function**.

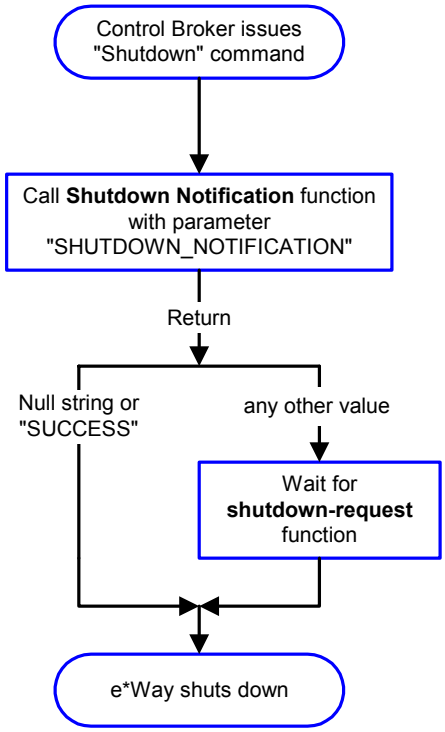
Figure 18 Disconnect Process



Shutdown Process

Figure 19 illustrates how the e*Way shuts itself down, using the **Shutdown Command Notification Function**.

Figure 19 Shutdown Process



Chapter 6

e*Way Configuration

This chapter describes how to use the e*Gate Enterprise Manager's e*Way Editor to prepare the Clarify e*Way for your specific implementation.

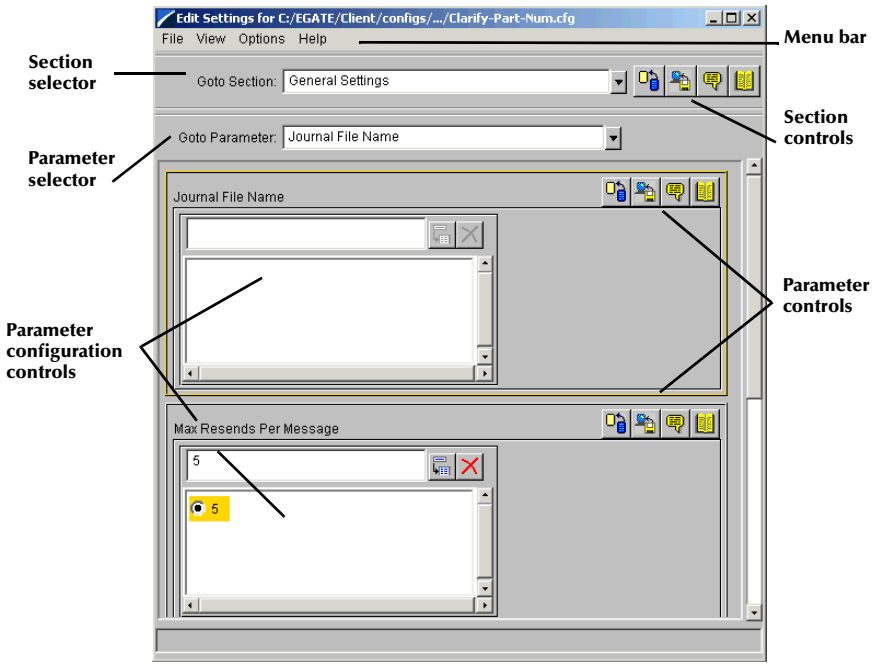
6.1 Using the e*Way Editor

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

6.1.1 User Interface

Figure 20 shows the e*Way Editor user interface.

Figure 20 The e*Way Editor



This interface has the following major features:

- The **Menu bar** provides access to basic operations: saving the configuration file, view a summary of all parameter settings, launching the Help system, and so on
- Configuration parameters are grouped into *sections*. The **Section selector** at the top of the Editor window enables you to select the section whose parameters you wish to edit
- **Section controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected section
- Each section contains a list of parameters. You can scroll through the entire list using the scroll bar, or use the **Parameter selector** to jump to a specific parameter
- **Parameter controls** enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected parameter (notice the similarity between the parameter controls and the section controls; the available operations are the same, but the scope differs)
- **Parameter configuration controls** enable you to set the e*Way's various operating parameters

Section and Parameter Controls

The section and parameter controls are shown in Table 5.

Table 5 Parameter and Section Controls

Button	Name	Function
	**Restore Default**	Restores default values
	**Restore Value**	Restores saved values
	**Tips**	Displays tips
	**User Notes**	Enters user notes

The controls next to the **Goto Section** selector affect all parameters in the selected section, whereas the controls within the parameter-editing section of the editor affect only the selected parameter.

Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

1. Option buttons (shown in Figure 20 on page 46 for setting the **AllowIncoming** and **AllowOutgoing** parameters).
2. Selection lists, as shown in Figure 21, which have controls as described in Table 6.

Figure 21 Sample Selection List Box

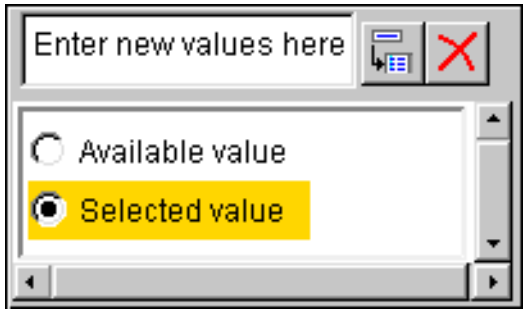


Table 6 Selection List Controls

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

6.1.2 Procedures

This section describes basic procedures you can use when working with the e*Way Editor. You may also view these procedures in the e*Way Editor’s online Help system.

Getting Help

To launch the e*Way Editor’s Help system

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

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

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

To display tips regarding a section of the e*Way configuration

On the **Section** Toolbar, click  .

To display tips regarding a parameter within the e*Way configuration

On the **Parameter** Toolbar, click  .

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

Navigating Through the Editor

To display a list of all e*Way parameters and their settings

1. From the **View** menu, select **Summary**.
2. Use the controls on the Summary list to go to a selected parameter or print a list of parameter settings.
3. When you have finished using the Summary list, click **Cancel**.

To view or edit a section of the e*Way configuration

In the **Goto Section** list, select a section.

To view or edit a selected parameter

1. In the **Goto Section** list, select the section that contains the parameter you want to edit.
2. In the **Goto Parameter** list, select a parameter. Alternatively, use the scroll bars to scroll down to the parameter.

To navigate using the summary list

1. From the **View** menu, select **Summary**.
2. Select a parameter and click **Go To**.

Saving Configuration Settings

To save the current configuration settings

Pull down the **File** menu and click **Save.**

Modifying Configuration Settings

To modify a configuration setting

1. Navigate to the parameter whose options you want to change.
2. Select the parameter from those that are provided. To add new options to a list of parameters (see **Figure 21 on page 48**), enter the option in the parameter's data-entry box and click  .
3. To remove options from the list, click  . A new dialog box displays, in which you perform the actual deletions.

Restoring Default Settings

To reload the default current configuration settings for a *parameter*

1 Navigate to the parameter whose defaults you want to restore.

2 On the **Parameter** toolbar, click  .

To reload the default current configuration settings for a *section*

1 Navigate to the section whose defaults you want to restore.

2 On the main toolbar, click  .

Restoring Saved Settings

To reload the saved current configuration settings for a *parameter*

1 Navigate to the parameter whose defaults you want to restore.

2 On the **Parameter** toolbar, click  .

To reload the saved current configuration settings for a *section*

1 Navigate to the section whose defaults you want to restore.

2 On the **Section** toolbar, click  .

Entering User Notes

User Notes provides you with a means of attaching comments to your e*Way configuration.

To enter user notes regarding the e*Way's general configuration or operations

From the **File** menu, select **User Notes.**

To enter user notes regarding a section of the e*Way configuration

On the **Section** toolbar, click  .

To enter user notes regarding a parameter within the e*Way configuration

On the **Parameter** toolbar, click  .

6.2 e*Way Configuration Parameters

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

To change e*Way configuration parameters

1. In the e*Gate Enterprise Manager'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.

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

General Settings on page 52

Communication Setup on page 53

Monk Configuration on page 55

Clarify Setup on page 62

6.2.1 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 are stored in the e*Gate **\SystemData** directory. See the *e*Gate Integrator System Administration and Operations Guide* for more information about file locations.

Additional Information

An Event is journaled for the following conditions:

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

Max Resends Per Message

Description

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

Required Values

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

Max Failed Messages

Description

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

Required Values

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

Forward External Errors

Description

Selects whether error messages that begin with the string **"DATAERR"** that are received from the external system are queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 58** for more information.

Required Values

Yes or **No**. The default value, **No**, specifies that error messages are not forwarded. See **"Basic e*Way Processes" on page 39** for more information about how the e*Way uses this function.

6.2.2 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 e*Gate Enterprise Manager controls when the e*Way executable runs. The schedule that you set within the parameters discussed in this section (using the e*Way Editor) determines when data is exchanged. Be sure you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

- 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 terminates 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 Acknowledgment Function** and **Negative Acknowledgment Function**) 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** is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

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

Exchange Data Interval

Description

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

Required Values

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

Additional Information

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

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

See **Down Timeout** on page 54 and **Stop Exchange Data Schedule** on page 54 for more information about the data-exchange schedule.

Down Timeout

Description

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

Required Values

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

Up Timeout

Description

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

Required Values

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

Resend Timeout

Description

Specifies the number of seconds the e*Way waits between attempts to resend 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**.

Zero Wait Between Successful Exchanges

Description

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

Required Values

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

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

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

6.2.3 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.

How to Specify Function Names or File Names

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

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

Additional Path

Description

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

Required Values

A 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 settings in the **.egate.store** file. See the *e*Gate Integrator System Administration and Operations Guide* for more information about this file.

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

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

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

Auxiliary Library Directories

Description

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

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

Required Values

A pathname, or a series of paths separated by semicolons.

Additional information

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

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

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

The default directory is **\monk_library**.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions to 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 is appended to the "load path." See **Additional Path** on page 55 for more information about the "load path."

Additional information

Any environment-initialization functions called by this file accept no input, and must return a string. The e*Way loads this file and tries to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way attempts 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 14 on page 40**).

Startup Function

Description

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

Required Values

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

Additional information

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

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

This function is called after the e*Way loads the specified **Monk Environment Initialization File** and any files within the specified **Auxiliary Directories**.

The e*Way loads this file and try to invoke a function of the same base name as the file name (see **Figure 14 on page 40**). For example, for a file named **my-startup.monk**, the e*Way attempts 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.

Note: *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 e*Gate Enterprise Manager). The function returns one of the following (see **Figure 16 on page 42** for more details):

- A 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 one of the preceding is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function

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

Exchange Data with External Function

Description

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

Required Values

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

Additional Information

The function accepts no input and must return a string (see **Figure 17 on page 44** for more details) as follows:

- A null string indicates that the data exchange was completed successfully; no information is sent into the e*Gate system.
- **CONNERR** indicates that a problem with the connection to the external system has occurred.
- **DATAERR** indicates that a problem with the data itself has occurred; the e*Way handles the string **DATAERR** (by itself), and **DATAERR** plus additional data, differently. See **Figure 17 on page 44** for more details.
- If a string other than one of the preceding is returned, the e*Way packages it 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 Exchange Data Schedule** or manually by the **start-schedule** Monk function. 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 immediately calls the **Exchange Data with External Function** again; otherwise, the e*Way does not call the function until the next scheduled "start exchange" time or the schedule is manually invoked using the Monk function **start-schedule**.

External Connection Establishment Function

Description

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

Required Values

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

Note: *You may* **not** *leave this field blank.*

Additional Information

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

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

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

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

External Connection Verification Function

Description

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

Required Values

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

Additional Information

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

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

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

The **External Connection Establishment Function** is called when the e*Way has determined that its connection to the external system is down.

External Connection Shutdown Function

Description

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

Note: *This parameter is* ***optional****.*

Required Values

The name of a Monk function.

Additional Information

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

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

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

Positive Acknowledgment Function

Description

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

Required Values

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

Additional Information

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

- **CONNERR** indicates a problem with the connection to the external system; when the connection is re-established, the **Positive Acknowledgment** function is called again (with the same input data).
- A null string indicates that the function executed successfully.

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

Negative Acknowledgment Function

Description

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

Required Values

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

Additional Information

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

- **CONNERR** indicates a problem with the connection to the external system; when the connection is re-established, the function is called again.
- A null string indicates that the function executed successfully.

This function is only called during the processing of inbound Events. After the **Exchange Data with External Function** returns a string that is transformed into an inbound Event, the Event is handed off to one or more Collaborations for further processing. If the Event's processing is not completed successfully by *all* the Collaborations to which it was sent, the e*Way executes the Negative Acknowledgment Function (otherwise, the e*Way executes the **Positive Acknowledgment Function**).

Shutdown Command Notification Function

Description

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

Note: *This parameter is* ***optional****.*

Required Values

The name of a Monk function.

Additional Information

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

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

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

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.*

6.2.4 Clarify Setup

Server Name

Description

Name of the Clarify server.

Required Values

A valid host name.

Default Value

None.

Database Name

Description

Name of the database to save to or query.

Required Values

A valid database file name.

Default Value

None.

User Name

Description

Name of the current user.

Required Values

A valid user name.

Default Value

None.

User Password

Description

Current user's password.

Required Values

A valid password.

Default Value

The default value is **0**.

Session SQL Dump?

Description

Flag used to indicate if SQL trace has been dumped.

Required Values

0:1

Default Value

The default value is **0**.

Retry Counts

Description

The allowed number of retrys.

Required Values

Integer value of the number of retries.

Default Value

The default value is **0**.

Retry Seconds

Description

Number of seconds to wait before retrying connection to Clarify database.

Required Values

Integer value of the number of seconds to wait.

Default Value

The default value is **0**.

Debug Log

Description

File name for the debug log.

Required Values

A valid file name.

Default Value

None.

Chapter 7

e*Way API Functions

This chapter describes the various Clarify API functions and structures, and Monk functions, used by the Clarify e*Way.

7.1 Overview

The Clarify e*Way uses the following list of APIs. If you are using Clarify 8 the **stc_monkclarify.dll** is used. If you are using Clarify 10.1, **stc_monkclarify_10.dll** is used for Windows and **stcewclarifymonk.exe** for UNIX operating systems.

The Clarify API functions are followed by the API structures. For additional information on each Clarify API function, see that function's structure.

7.1.1 Clarify API Functions and Structures

Clarify API functions and structures are categorized as:

Clarify High-level API Functions on page 66

Clarify High-level API Structures on page 107

Clarify API (API Toolkit) Functions on page 122

Clarify API (API Toolkit) Structures on page 167

Miscellaneous Functions and Structures on page 174

7.1.2 Clarify e*Way Monk Functions

Architecturally, the e*Way can be viewed as a two-layered structure, consisting of:

- Clarify Transport Layer
- e*Way Kernel Layer

Each layer contains Monk scripts and/or functions, and makes use of lower-level Monk functions residing in the layer beneath. You, as user, primarily make use of the highest-level functions, which reside in the Clarify Transport Layer.

The Clarify e*Way's Monk functions can be categorized as:

Clarify Monk Functions on page 175

Basic Functions on page 193

7.2 Clarify High-level API Functions

Create New Case

zca_new_create on page 68

zca_new_reset on page 69

zca_new_simple_find_all on page 70

zca_new_find_contact on page 71

zca_new_find_contract on page 72

zca_new_find_site on page 73

zca_new_find_site_part on page 74

zca_new_execute on page 75

zca_new_destroy on page 76

Dispatch Object

zqu_dsp_create on page 77

zqu_dsp_reset on page 78

zqu_dsp_find_available_queues on page 79

zqu_dsp_execute on page 80

zqu_dsp_destroy on page 81

Log Notes on Case

zac_pad_create on page 82

zac_pad_reset on page 83

zac_pad_execute on page 84

zac_pad_destroy on page 85

Change Case Status

zac_sts_create on page 86

zac_sts_reset on page 87

zac_sts_execute on page 88

zac_sts_destroy on page 89

Create Subcase

zsb_new_create on page 90

zca_new_create

Description

Allocates **zca_ty_cas_struct** data structure and initializes the data structure with null values.

Syntax

```
(zca_new_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zca_ty_cas_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define new-case (zca_new_create 0))
```

zca_new_reset

Description

Initializes the **zca_ty_cas_struct** data structure.

Syntax

```
(zca_new_reset <dbid-ptr> <case-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid *	Database ID pointer.
case-ptr	zca_ty_cas_struct *	Case pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid (cpi_sess_login "CLFY" "CLFY" "User" "Passwd"))
(define new-case (zca_new_create 0))
(zca_new_reset dbid new-case 0)
```

zca_new_simple_find_all

Description

Obtains object handles from the database, based on data loaded in the **zca_ty_simple_find** data structure, and places the handles into the **zca_ty_cas_struct** data structure.

Syntax

```
(zca_new_simple_find_all <simple_find> <0> <case_ptr>)
```

Parameters

Name	Type	Description
simple_find	zca_ty_simple_find *	Source data structure.
0	long int	Routine option flags.
case_ptr	zca_ty_cas_struct *	Case pointer.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define simple-find (get_zca_ty_simple_find_ptr "LastName"
"FirstName" "555-1212" "" "" "" "" "" ""))
(define new-case (zca_new_create 0))
(zca_new_simple_find_all simple-find 0 new-case)
```

zca_new_find_contact

Description

Finds a list of contact objects that matches with criteria, based on **cpi_ty_cond <condition>** or **cpi_ty_obj <object_list>**.

Syntax

```
(zca_new_find_contact <case_ptr> <object_list> <count>
    <condition> <0>)
```

Parameters

Name	Type	Description
case_ptr	zca_ty_cas_struct	Case pointer.
object_list	cpi_ty_obj	Object list.
count	long int	Number of objects in list.
condition	cpi_ty_cond	Specified condition.
0	long int	Routine option flags.

Returns

Upon successful execution, a **vector** of object handles (long int); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define null-obj-list (get_cpi_ty_obj_null_ptr))
(define new-case (zca_new_create 0))
(define fcond-ptr (get_cpi_ty_fcond_ptr `#(#(4 "part_num" "PART"))))
(define cond-ptr (get_cpi_ty_cond_ptr 1 fcond-ptr 1))
(zca_new_find_contact new-case null-obj-list 0 cond-ptr 0)
```

zca_new_find_contract

Description

Finds a list of contact objects that matches with criteria, based on **cpi_ty_cond <condition>** or **cpi_ty_obj <object_list>**.

Syntax

```
(zca_new_find_contract <case_ptr> <object_list> <count> <condition>
<0>)
```

Parameters

Name	Type	Description
case_ptr	zca_ty_cas_struct	Case pointer.
object_list	cpi_ty_obj	Object list.
count	long int	Number of objects in list.
condition	cpi_ty_cond	Specified condition.
0	long int	Routine option flags.

Returns

Upon successful execution, a **vector** of object handles (long int); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define null-obj-list (get_cpi_ty_obj_null_ptr))
(define new-case (zca_new_create 0))
(define fcond-ptr (get_cpi_ty_fcond_ptr '#(#(4 "part_num" "PART"))))
(define cond-ptr (get_cpi_ty_cond_ptr 1 fcond-ptr 1))
(zca_new_find_contract new-case null-obj-list 0 cond-ptr 0)
```

zca_new_find_site

Description

Finds a list of site objects that matches with criteria, based on **cpi_ty_cond <condition>** or **cpi_ty_obj <object_list>**.

Syntax

```
(zca_new_find_site <case_ptr> <object_list> <count> <condition> <0>)
```

Parameters

Name	Type	Description
case_ptr	zca_ty_cas_struct	Case pointer.
object_list	cpi_ty_obj	Object list.
count	long int	Number of objects.
condition	cpi_ty_cond	Condition.
0	long int	Routine option flags.

Returns

Upon successful execution, a **vector** of object handles (long int); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-list (get_cpi_ty_obj_ptr '#(2345672 3121433)))
(define new-case (zca_new_create 0))
(define null-cond (get_cpi_ty_cond_null_ptr))
(zca_new_find_site new-case obj-list 0 null-cond 0)
```

zca_new_find_site_part

Description

Finds a list of **site_part** objects that matches with criteria, based on **cpi_ty_cond <condition>** or **cpi_ty_obj <object_list>**.

Syntax

```
(zca_new_find_site_part <case_ptr> <object_list> <count> <condition>
<0>)
```

Parameters

Name	Type	Description
case_ptr	zca_ty_cas_struct	Case pointer.
object_list	cpi_ty_obj	Object list.
count	long int	Number of objects.
condition	cpi_ty_cond	Condition.
0	long int	Routine option flags.

Returns

Upon successful execution, a **vector** of object handles (long int); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-list (get_cpi_ty_obj_ptr '#(2345672 3121433)))
(define new-case (zca_new_create 0))
(define null-cond (get_cpi_ty_cond_null_ptr))
(zca_new_find_site_part new-case obj-list 0 null-cond 0)
```

zca_new_execute

Description

Executes the new case operation, using information stored in **case_ptr**.

Syntax

```
(zca_new_execute <case_ptr> <0>)
```

Parameters

Name	Type	Description
case_ptr	zca_ty_cas_struct	Case pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define new-case (zca_new_create 0))
.
.
.
(zca_new_execute new-case 0)
```

zca_new_destroy

Description

De-allocates memory used by **case_ptr**.

Syntax

```
(zca_new_destroy <case_ptr> <0>)
```

Parameters

Name	Type	Description
case_ptr	zca_ty_cas_struct	Case pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define new-case (zca_new_create 0))
.
.
.
(zca_new_destroy new-case 0)
```

zqu_dsp_create

Description

Allocates **alloc_struct** data structure and initializes the data structure with null values

Syntax

```
(zqu_dsp_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zqu_ty_dsp_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dispatch-ptr (zqu_dsp_create 0))
```

zqu_dsp_reset

Description

Accepts filter conditions **parent_cond_ptr** for the case (or subcase) and filter conditions **queue_cond_ptr** for the target queue and uses them to return the object handles of the case (or subcase) and queue. These handles are placed in the dispatch structure **zqu_ty_dsp_struct**. Notice that only one case/subcase object handle and one queue handle is returned to the data structure. If you want a list of queue handles, call **zqu_dsp_find_available_queues** after calling **reset**.

Syntax

```
(zqu_dsp_reset <dbid-ptr> <dispatch-ptr> <0> <parent-cond-ptr>
<queue-cond-ptr>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
dispatch-ptr	zqu_ty_dsp_struct	Dispatch pointer.
0	long int	Routine option flags.
parent-cond-ptr	zpi_ty_cond	Parent condition pointer.
queue-cond-ptr	zpi_ty_cond	Queue condition pointer.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid (cpi_sess_login "CLFY" "CLFY" "User" "Passwd"))
(define dispatch-ptr  (zqu_dsp_create 0))
(define null-parent-cond (get_cpi_ty_cond_null_ptr))
(define null-queue-cond (get_cpi_ty_cond_null_ptr))
(zqu_dsp_reset dbid null-parent-cond null-queue-cond 0 dispatch-ptr)
```

zqu_dsp_find_available_queues

Description

Returns list of queue object handles.

Syntax

```
(zqu_dsp_find_available_queues <dbid-ptr> <state> <case-obj-handle>
<obj-type> <0> <condition>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database pointer.
state	long int	State.
case-obj-handle	long int	Case object handle.
obj-type	char	Object type.
0	long int	Routine option flags.
condition	cpi_ty_cond	Condition.

Returns

A **vector** of object handles (long int) on success; otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid (cpi_sess_login "CLFY" "CLFY" "User" "Passwd"))
(deifne tstate (cpi_state_create 0))
(define null-cond (get_cpi_ty_cond_null_ptr))
(define obj-vec (zqu_dsp_find_available_queues dbid tstate 1234567
"object-type" 0 null-cond))
```

zqu_dsp_execute

Description

Executes the dispatch operation, using information stored in **dispatch-ptr.**

Syntax

```
(zqu_dsp_execute <dispatch-ptr> <0>)
```

Parameters

Name	Type	Description
dispatch-ptr	zqu_ty_dsp_struct	Dispatch pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dispatch-ptr (zqu_dsp_create 0))
.
.
.
(zqu_dsp_execute dispatch-ptr 0)
```

zqu_dsp_destroy

Description

De-allocates memory used by **dispatch-ptr.**

Syntax

```
(zqu_dsp_destroy <dispatch-ptr> <0>)
```

Parameters

Name	Type	Description
dispatch-ptr	zqu_ty_dsp_struct	Dispatch pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dispatch-ptr (zqu_dsp_create 0))
.
.
.
(zqu_dsp_destroy dispatch-ptr 0)
```

zac_pad_create

Description

Allocates **zac_ty_log_struct** data structure and initializes the data structure with null values.

Syntax

```
(zac_pad_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zac_ty_log_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define log-ptr (zac_pad_create 0))
```

zac_pad_reset

Description

Accepts case filter conditions from a **zpi_ty_cond** structure and returns the appropriate case object handle from the database: the handle is placed into the log notes structure.

Syntax

```
(zac_pad_reset <dbid-ptr> <parent-cond> <log-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database pointer.
parent-cond	zpi_ty_cond	Parent condition.
log-ptr	zac_ty_log_struct	Log pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" User" "Password"))
(define null-cond (get_zpi_ty_cond_null_ptr))
(define log-ptr (zac_pad_create 0))
(zac_pad_reset dbid-ptr null-cond 0 log-ptr)
```

zac_pad_execute

Description

Executes the dispatch operation, using information stored in **log-ptr**.

Syntax

```
(zac_pad_execute <log-ptr> <0>)
```

Parameters

Name	Type	Description
log-ptr	zac_ty_log_struct	Log pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define log-ptr (zac_pad_create 0))
.
.
.
(zac_pad_execute log-ptr 0)
```

zac_pad_destroy

Description

De-allocates memory used by **log-ptr**.

Syntax

```
(zac_pad_destroy <log-ptr> <0>)
```

Parameters

Name	Type	Description
log-ptr	zac_ty_log_struct	Log pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define log-ptr (zac_pad_create 0))
.
.
.
(zac_pad_destroy log-ptr 0)
```

zac_sts_create

Description

Allocates **zac_ty_sts_struct** data structure and initializes the data structure with null values.

Syntax

```
(zac_sts_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zac_ty_sts_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define sts-ptr (zac_sts_create 0))
```

zac_sts_reset

Description

Accepts filter conditions from a **zpi_ty_cond** structure and a new status that you supply from the command line or from some other source. It returns the parent case object handle and the new status to the to the change status structure **zac_ty_sts_struct**.

Syntax

```
(zac_sts_reset <dbid-ptr> <obj-cond> <new-sts-name> <0> <sts-ptr>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
obj-cond	zpi_ty_cond	Object condition.
new-sts-name	char	New status name.
0	long int	Routine option flags.
sts-ptr	zac_ty_sts_struct	Status pointer.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" User" "Password"))
(define null-cond (get_zpi_ty_cond_null_ptr))
(define sts-ptr (zac_sts_create 0))
(zac_sts_reset dbid-ptr null-cond "New-STS-Name" 0 sts-ptr)
```

zac_sts_execute

Description

Executes the dispatch operation, using information stored in **sts-ptr**.

Syntax

```
(zac_sts_execute <sts-ptr> <0>)
```

Parameters

Name	Type	Description
sts-ptr	zac_ty_sts_struct	Status pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define sts-ptr (zac_sts_create 0))
.
.
.
(zac_sts_execute sts-ptr 0)
```

zac_sts_destroy

Description

De-allocates memory used by **sts-ptr**.

Syntax

```
(zac_sts_destroy <sts-ptr> <0>)
```

Parameters

Name	Type	Description
sts-ptr	zac_ty_sts_struct	Status pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define sts-ptr (zac_sts_create 0))
.
.
.
(zac_sts_destroy sts-ptr 0)
```

zsb_new_create

Description

Allocates **zac_ty_sts_struct** data structure and initializes the data structure with null values

Syntax

```
(zsb_new_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zsb_ty_subcase_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define subcase-ptr (zsb_new_create 0))
```

zsb_new_reset

Description

Creates the required database objects for creating a subcase and fills the subcase object with default values. Also checks whether the passed in case is open; if not open, the operation fails.

Syntax

```
(zsb_new_reset <dbid-ptr> <subcase-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
subcase-ptr	zsb_ty_subcase_struct	Subcase pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" User" "Password"))
(define subcase-ptr (zsb_new_create 0))
(zsb_new_reset dbid-ptr subcase-ptr 0)
```

zsb_new_set_reqd_date

Description

By default, the required date field in the subcase is set to the current date plus one day. To specify a different required date, you must call this function subsequent to the call to **reset**.

Syntax

```
(zsb_new_set_reqd_date <subcase-ptr> <hours> <days>)
```

Parameters

Name	Type	Description
subcase-ptr	zsb_ty_subcase_struct	Subcase pointer.
hours	long int	Reset to hours.
days	long int	Reset to days.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" User" "Password"))
(zsb_new_set_reqd_date dbid-ptr 23 30)
```

Note: *This function is optional.*

zsb_new_execute

Description

Executes the dispatch operation, using information stored in **subcase-ptr**.

Syntax

```
(zsb_new_execute <subcase-ptr> <0>)
```

Parameters

Name	Type	Description
subcase-ptr	zsb_ty_subcase_struct	Subcase pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define subcase-ptr (zsb_new_create 0))
.
.
.
(zsb_new_execute subcase-ptr 0)
```

zsb_new_destroy

Description

De-allocates memory used by **subcase-ptr**.

Syntax

```
(zsb_new_destroy <subcase-ptr> <0>)
```

Parameters

Name	Type	Description
subcase-ptr	zsb_ty_subcase_struct	Subcase pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define subcase-ptr (zsb_new_create 0))
.
.
.
(zsb_new_destroy subcase-ptr 0)
```

zrq_hdr_create

Description

Allocates **zrq_ty_hdr_struct** data structure and initializes the data structure with null values.

Syntax

```
(zrq_hdr_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zrq_ty_hdr_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define hdr-ptr (zrq_hdr_create 0))
```

zrq_hdr_reset

Description

Called subsequent to the call to **zrq_hdr_create**. It sets creates the necessary database objects and sets default values for certain fields in the header. Notice that you do not set any values in the header (except for the required value, either case object handle or site object handle) until after you call **zrq_hdr_reset**.

Syntax

```
(zrq_hdr_reset <dbid-ptr> <obj-cond> <hdr-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
obj-cond	zpi_ty_cond	Object condition.
hdr-ptr	zrq_ty_hdr_struct	Header pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" User" "Password"))
(define null-cond (get_zpi_ty_cond_null_ptr))
(define hdr-ptr (zrq_hdr_create 0))
(zrq_hdr_reset dbid-ptr null-cond hdr-ptr 0)
```

zrq_hdr_simple_find

Description

Fills in three fields of the part request header: **priority, pay_method**, and **pay_terms**. By default, these fields in the header have the values set by the **zrq_hdr_reset** function. To specify a different values, this function must be called subsequent to the call to **zrq_hdr_reset**. Note also that in C API, arg1-arg3 are fields in **zrq_hdr_simple_find** strcuture as used in the example.

Syntax

```
(zrq_hdr_simple_find <priority> <payment-method> <payterm> <hdr-ptr>
<0>)
```

Parameters

Name	Type	Description
priority	char	Priority.
payment-method	char	Payment method.
payterm	char	Payment term.
hdr-ptr	zrq_ty_hdr_struct	Header pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define hdr-ptr (zrq_hdr_create 0))
(zrq_hdr_simple_find "high" "payment-method" "payment term" hdr-ptr
0)
```

Note: *This function is optional.*

zrq_hdr_execute

Description

Executes the dispatch operation, using information stored in **hdr-ptr**.

Syntax

```
(zrq_hdr_execute <hdr-ptr> <0>)
```

Parameters

Name	Type	Description
hdr-ptr	zrq_ty_hdr_struct	Header pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define hdr-ptr (zrq_hdr_create 0))
.
.
.
(zrq_hdr_execute hdr-ptr 0)
```

zrq_hdr_destroy

Description

De-allocates memory used by **hdr-ptr**.

Syntax

```
(zrq_hdr_destroy <hdr-ptr> <0>)
```

Parameters

Name	Type	Description
hdr-ptr	zrq_ty_hdr_struct	Header pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define hdr-ptr (zrq_hdr_create 0))
.
.
.
(zrq_hdr_destroy hdr-ptr 0)
```

zrq_dtl_create

Description

Allocates **zrq_ty_dtl_struct** data structure and initializes the data structure with null values.

Syntax

```
(zrq_dtl_create <0>)
```

Parameters

Name	Type	Description
0	long int	Routine option flags.

Returns

Upon successful execution, a Monk object of **zrq_ty_dtl_struct** custom type; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dtl-ptr (zrq_dtl_create 0))
```

zrq_dtl_reset

Description

Called subsequent to the call to **zrq_dtl_create**. It initializes creates the required database objects, creates required relations, and fills default field values in the detail object.

Syntax

```
(zrq_dtl_reset <dbid-ptr> <obj-cond> <part-num> <mod-level> <count>
<dtl-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
obj-cond	zpi_ty_cond	Object condition.
part-num	char	Part number.
mod-level	char	Modification level.
count	long int	Number of objects.
dtl-ptr	zrq_ty_dtl_struct	Detail pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" User" "Password"))
(define null-cond (get_zpi_ty_cond_null_ptr))
(define dtl-ptr (zrq_dtl_create 0))
(zrq_dtl_reset dbid-ptr null-cond "PART" "MOD" 1 dtl-ptr 0)
```

zrq_dtl_simple_find

Description

Sets certain fields in the part request detail; for inventory parts (that is, parts from the Parts List). For site parts installed at a particular site (that is, parts from the Site Configuration List), this function is required to specify that the part is a site part. In C API, arg1-arg11 are part of **zrq_dtl_simple_find** structure as used in the example.

Syntax

```
(zrq_dtl_simple_find <hdr-id> <bin-name> <mod-level> <part-num>
<serial-number> <is-site-part> <request> <type> <priority po-number>
<ship-via> <detail-notes> <dtl-ptr> <0>)
```

Parameters

Name	Type	Description
hdr-id	char	Header ID.
bin-name	char	Bin (location) name.
mod-level	char	Modification level.
part-num	char	Part number.
serial-number	char	Serial number.
is-site-part	long int	Site part indicator.
request type	char	Request type.
priority	char	Priority.
po-number	char	Purchase order number.
ship-via	char	Shipping method.
detail-notes	char	Detail notes.
dtl-ptr	zra_ty_dtl_struct	Detail pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define null-find (get_zrq_ty_find_null_ptr))
```

```
(define dtl-ptr (zrq_dtl_create 0))
(zrq_dtl_simple_find null-find 0 dtl-ptr)
```

Note: *This function is optional.*

zrq_dtl_execute

Description

Executes the dispatch operation, using information stored in **dtl-ptr.**

Syntax

```
(zrq_dtl_execute <dtl-ptr> <0>)
```

Parameters

Name	Type	Description
dtl-ptr	zrq_ty_dtl_struct	Detail pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dtl-ptr (zrq_dtl_create 0))
.
.
.
(zrq_dtl_execute dtl-ptr 0)
```

zrq_dtl_destroy

Description

De-allocates memory used by **dtl-ptr**.

Syntax

```
(zrq_dtl_destroy <dtl-ptr> <0>)
```

Parameters

Name	Type	Description
dtl-ptr	zrq_ty_dtl_struct	Detail pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dtl-ptr (zrq_dtl_create 0))
.
.
.
(zrq_dtl_destroy dtl-ptr 0)
```

zpi_db_start_batch

Description

Used to keep the data created by **HLAPI** functions in local memory as much as possible to minimize network round trips. After this function is invoked, most batched-up data is kept locally until a call to **zpi_db_execute_batch** is made.

Syntax

```
(zpi_db_start_batch <dbid-ptr>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "User" "Password"))
(zpi_db_start_batch dbid-ptr)
```

zpi_db_execute_batch

Description

Commits batched-up data to the database. It is used in conjunction with the **zpi_db_start_batch** function.

Syntax

```
(zpi_db_execute_batch <dbid-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "User" "Password"))
(zpi_db_execute_batch dbid-ptr 0)
```

7.3 Clarify High-level API Structures

Create New Case

zca_ty_cas_struct on page 109

zca_ty_simple_find on page 111

Dispatch Object

zqu_ty_dsp_struct on page 112

zpi_ty_cond on page 113

Log Notes on Case

zac_ty_log_struct on page 114

Change Case Status

zac_ty_sts_struct on page 116

Create Subcase

zsb_ty_subcase_struct on page 118

Create Part Request Header

zrq_ty_hdr_struct on page 120

Create Part Request Detail

zrq_ty_dtl_struct on page 121

zca_ty_cas_struct

Structure Definition

```
typedef struct
    {
    cpi_ty_obj          case_obj_hdl;             /* REQUIRED, NEW */
    cpi_ty_obj          status_obj_hdl;           /* OPTIONAL, NEW */
    cpi_ty_obj          phone_log_obj_hdl;        /* OPTIONAL, NEW */
    cpi_ty_obj          contact_obj_hdl;          /* REQUIRED      */
    cpi_ty_obj          contract_obj_hdl;         /* OPTIONAL      */
    cpi_ty_obj          site_obj_hdl;             /* OPTIONAL      */
    cpi_ty_obj          wbin_obj_hdl;             /* OPTIONAL      */
    cpi_ty_obj          state_obj_hdl;            /* OPTIONAL      */
    cpi_ty_obj          calltype_obj_hdl;         /* OPTIONAL      */
    cpi_ty_obj          severity_obj_hdl;         /* OPTIONAL      */
    cpi_ty_obj          priority_obj_hdl;         /* OPTIONAL      */
    cpi_ty_obj          prod_inst_hdl;            /* OPTIONAL      */
    cpi_ty_obj          de_prod_obj_hdl;          /* OPTIONAL      */
    cpi_ty_obj          covrd_ppi_prod_obj_hdl;   /* OPTIONAL      */
    cpi_ty_obj          pnum_obj_hdl;             /* OPTIONAL      */
    cpi_ty_obj          de_pnum_obj_hdl;          /* OPTIONAL      */
    cpi_ty_obj          user_obj_hdl;             /* OPTIONAL      */
    cpi_ty_obj          address_obj_hdl;          /* OPTIONAL      */
    long int            user_name_max_length;     /* max. chars    */
    char               *user_name;                /* do not free   */
    long int            case_id_max_length;       /* max. chars    */
    char               *case_id;                  /* do not free   */
    }
zca_ty_new_case_part;

typedef struct
    {
    cpi_ty_state   tstate;
    long int       flags;
    void          *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part; /* PROTECTED DATA */
    zca_ty_new_case_part new_case_part;
    }
zca_ty_cas_struct;
```

Type Creation Functions

```
(define case-ptr (zca_new_create dbid-ptr 0))
(define null-case-ptr (get_zca_ty_cas_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zca_ty_cas_struct? case-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zca_ty_cas_struct_int_get_case_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_status_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_phone_log_obj_hdl case-
ptr))
(define int-ret (zca_ty_cas_struct_int_get_contact_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_contract_obj_hdl case-
ptr))
(define int-ret (zca_ty_cas_struct_int_get_site_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_wbin_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_state_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_calltype_obj_hdl case-
ptr))
(define int-ret (zca_ty_cas_struct_int_get_severity_obj_hdl case-
ptr))
(define int-ret (zca_ty_cas_struct_int_get_priority_obj_hdl case-
ptr))
(define int-ret (zca_ty_cas_struct_int_get_prod_inst_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_de_prod_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_covrd_ppi_prod_obj_hdl
case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_pnum_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_de_pnum_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_user_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_address_obj_hdl case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_user_name_max_length case-
ptr))
(define str-ret (zca_ty_cas_struct_str_get_user_name case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_case_id_max_length case-
ptr))
(define str-ret (zca_ty_cas_struct_str_get_case_id case-ptr))
(define int-ret (zca_ty_cas_struct_int_get_tstate case-ptr))
```

Field-accessing Functions (set)

```
(zca_ty_cas_struct_int_set_case_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_status_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_phone_log_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_contact_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_contract_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_site_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_wbin_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_state_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_calltype_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_severity_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_priority_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_prod_inst_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_de_prod_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_covrd_ppi_prod_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_pnum_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_de_pnum_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_user_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_address_obj_hdl case-ptr int-value)
(zca_ty_cas_struct_int_set_user_name_max_length case-ptr int-value)
(zca_ty_cas_struct_str_set_user_name case-ptr str-value)
(zca_ty_cas_struct_int_set_case_id_max_length case-ptr int-value)
(zca_ty_cas_struct_str_set_case_id case-ptr str-value)
(zca_ty_cas_struct_int_set_tstate case-ptr int-value)
```

zca_ty_simple_find

Structure Definition

```
typedef struct
    {
    char*call_type;
    char*response_priority;
    char*response_severity;
    char*serial_no;
    char*site_id;
    char*contact_last_name;
    char*contact_first_name;
    char*contact_phone;
    char*contract_id;
    }
zca_ty_simple_find;
```

Type Creation Functions

```
(define simple-find-ptr (get_zca_ty_simple_find_ptr
last-name
first-name
contact-phone
call-type
response-priority
response-severity
serial-number
site-id
contract-id)
(define null-simple-find (get_zca_ty_simple_find_null_ptr))
```

Type Validation Function

```
(define bool-ret (zca_ty_simple_find_ptr? simple-find-ptr))
```

zqu_ty_dsp_struct

Structure Definition

```
typedef struct
    {
    cpi_ty_obj obj_hdl;    /* REQUIRED object to be dispatched*/
    cpi_ty_obj queue_obj_hdl;             /* REQUIRED */
    cpi_ty_obj wip_obj_hdl;               /* optional */
    cpi_ty_obj state_obj_hdl;             /* optional */
    cpi_ty_obj user_obj_hdl;              /* optional */
    cpi_ty_obj owner_obj_hdl;             /* optional */
    char obj_type[CPI_MAX_OBJECT_TYPE];   /* optional */
    }
zqu_ty_dispatch_part;

typedef struct
    {
    cpi_ty_state tstate;
    long int flags;
    void *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part;  /* PROTECTED DATA */
    zqu_ty_dispatch_part dispatch_part;
    }
zqu_ty_dsp_struct;
```

Type Creation Functions

```
(define dispatch-ptr (zqu_dsp_create dbid-ptr 0))
(define null-dispatch-ptr (get_zqu_ty_dsp_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zqu_ty_dsp_struct? dispatch-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zqu_ty_dsp_struct_int_get_obj_hdl dispatch-ptr))
(define int-ret (zqu_ty_dsp_struct_int_get_queue_obj_hdl dispatch-
ptr))
(define int-ret (zqu_ty_dsp_struct_int_get_wip_obj_hdl dispatch-ptr))
(define int-ret (zqu_ty_dsp_struct_int_get_state_obj_hdl dispatch-
ptr))
(define int-ret (zqu_ty_dsp_struct_int_get_user_obj_hdl dispatch-
ptr))
(define int-ret (zqu_ty_dsp_struct_int_get_owner_obj_hdl dispatch-
ptr))
(define str-ret (zqu_ty_dsp_struct_str_get_obj_type dispatch-ptr))
(define int-ret (zqu_ty_dsp_struct_int_get_tstate dispatch-ptr))
```

Field-accessing Functions (set)

```
(zqu_ty_dsp_struct_int_set_obj_hdl dispatch-ptr int-value)
(zqu_ty_dsp_struct_int_set_queue_obj_hdl dispatch-ptr int-value)
(zqu_ty_dsp_struct_int_set_wip_obj_hdl dispatch-ptr int-value)
(zqu_ty_dsp_struct_int_set_state_obj_hdl dispatch-ptr int-value)
(zqu_ty_dsp_struct_int_set_user_obj_hdl dispatch-ptr int-value)
(zqu_ty_dsp_struct_int_set_owner_obj_hdl dispatch-ptr int-value)
(zqu_ty_dsp_struct_str_set_obj_type dispatch-ptr str-value)
(zqu_ty_dsp_struct_int_set_tstate dispatch-ptr int-value)
```

zpi_ty_cond

Structure Definition

```
typedef struct
    {
    cpi_ty_obj obj_handle;
    cpi_ty_cond selection_cond;
    char obj_type[CPI_MAX_OBJECT_TYPE];
    }
zpi_ty_cond;
```

Type Creation Functions

```
(define zpi-cond-ptr (get_zpi_ty_cond_ptr obj-handle cpi-cond-ptr
obj-type)
(define null-zpi-cond-ptr (get_zpi_ty_cond_null_ptr))
```

Type Validation Function

```
(define bool-ret (zpi_ty_cond_ptr? zpi-cond-ptr))
```

zac_ty_log_struct

Structure Definition

```
typedef struct
    {
    cpi_ty_obj grand_parent_hdl;          /* OPTIONAL */
    cpi_ty_obj parent_obj_hdl;            /* REQUIRED */
    cpi_ty_obj log_obj_hdl;               /* REQUIRED */
    cpi_ty_obj condition_obj_hdl;         /* OPTIONAL */
    cpi_ty_obj user_obj_hdl;              /* OPTIONAL */
    cpi_ty_obj owner_obj_hdl;             /* OPTIONAL */
    cpi_ty_obj cur_contact_obj_hdl;       /* OPTIONAL */
    cpi_ty_obj new_contact_obj_hdl;   /* new_contact object for GUI
                                                 */
    char obj_type[CPI_MAX_OBJECT_TYPE];   /* OPTIONAL */
    }
zac_ty_log_notes_part;

typedef struct
    {
    cpi_ty_state tstate;
    long int flags;
    void *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part;/* PROTECTED DATA */
    zac_ty_log_notes_part log_notes_part;
    }
zac_ty_log_struct;
```

Type Creation Functions

```
(define log-ptr (zac_pad_create dbid-ptr 0))
(define null-log-ptr (get_zac_ty_log_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zac_ty_log_struct? log-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zac-ty_log_struct_int_get_grand_parent_hdl log-ptr))
(define int-ret (zac-ty_log_struct_int_get_parent_obj_hdl log-ptr))
(define int-ret (zac-ty_log_struct_int_get_log_obj_hdl log-ptr))
(define int-ret (zac-ty_log_struct_int_get_condition_obj_hdl log-
ptr))
(define int-ret (zac-ty_log_struct_int_get_user_obj_hdl log-ptr))
(define int-ret (zac-ty_log_struct_int_get_owner_obj_hdl log-ptr))
(define int-ret (zac-ty_log_struct_int_get_cur_contact_obj_hdl log-
ptr))
(define int-ret (zac-ty_log_struct_int_get_new_contact_obj_hdl log-
ptr))
(define str-ret (zac-ty_log_struct_str_get_obj_type log-ptr))
(define int-ret (zac-ty_log_struct_int_get_tstate log-ptr))
```

Field-accessing Functions (set)

```
(zac_ty_log_struct_int_set_grand_parent_hdl log-ptr int-value)
(zac_ty_log_struct_int_set_parent_obj_hdl log-ptr int-value)
(zac_ty_log_struct_int_set_log_obj_hdl log-ptr int-value)
(zac_ty_log_struct_int_set_condition_obj_hdl log-ptr int-value)
(zac_ty_log_struct_int_set_user_obj_hdl log-ptr int-value)
```

```
(zac_ty_log_struct_int_set_owner_obj_hdl log-ptr int-value)
(zac_ty_log_struct_int_set_cur_contact_obj_hdl log-ptr int-value)
(zac_ty_log_struct_int_set_new_contact_obj_hdl log-ptr int-value)
(zac_ty_log_struct_str_set_obj_type log-ptr int-value)
(zac_ty_log_struct_int_set_tstate log-ptr int-value)
```

zac_ty_sts_struct

Structure Definition

```
typedef struct
    {
    cpi_ty_obj grand_parent_hdl;          /* OPTIONAL */
    cpi_ty_obj parent_obj_hdl;            /* REQUIRED */
    cpi_ty_obj status_chg_obj_hdl;        /* OPTIONAL */
    cpi_ty_obj condition_obj_hdl;         /* OPTIONAL */
    cpi_ty_obj user_obj_hdl;              /* OPTIONAL */
    cpi_ty_obj new_gse_obj_hdl;           /* REQUIRED */
    cpi_ty_obj old_gse_obj_hdl;           /* OPTIONAL */
    cpi_ty_obj act_entry_obj_hdl;         /* OPTIONAL */
    cpi_ty_obj owner_obj_hdl;             /* OPTIONAL */
    cpi_ty_col status_col;                /* OPTIONAL */
    char obj_type[CPI_MAX_OBJECT_TYPE];   /* OPTIONAL */
    }
zac_ty_chg_sts_part;

typedef struct
    {
    cpi_ty_state tstate;
    long int flags;
    void *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part; /* PROTECTED DATA */
    zac_ty_chg_sts_part chg_sts_part;/* Use this structure */
    }
zac_ty_sts_struct;
```

Type Creation Functions

```
(define sts-ptr (zac_sts_create dbid-ptr 0))
(define null-sts-ptr (get_zac_ty_sts_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zac_ty_sts_struct? sts-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zac_ty_sts_struct_int_get_grand_parent_hdl sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_parent_obj_hdl sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_status_chg_obj_hdl sts-
ptr))
(define int-ret (zac_ty_sts_struct_int_get_condition_obj_hdl sts-
ptr))
(define int-ret (zac_ty_sts_struct_int_get_user_obj_hdl sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_new_gse_obj_hdl sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_old_gse_obj_hdl sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_act_entry_obj_hdl sts-
ptr))
(define int-ret (zac_ty_sts_struct_int_get_owner_obj_hdl sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_status_col sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_obj_type sts-ptr))
(define int-ret (zac_ty_sts_struct_int_get_tstate sts-ptr))
```

Field-accessing Functions (set)

```
(zac_ty_sts_struct_int_set_grand_parent_hdl sts-ptr)
(zac_ty_sts_struct_int_set_parent_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_status_chg_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_condition_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_user_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_new_gse_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_old_gse_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_act_entry_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_owner_obj_hdl sts-ptr int-value)
(zac_ty_sts_struct_int_set_status_col sts-ptr int-value)
(zac_ty_sts_struct_int_set_obj_type sts-ptr int-value)
(zac_ty_sts_struct_int_set_tstate sts-ptr int-value)
```

zsb_ty_subcase_struct

Structure Definition

```
typedef struct
    {
                /* User related information. */
    long int options;                   /* RESERVED for future */
    long int user_name_max_length;/* max length for user name */
    char *user_name;                    /* set by caller */
    cpi_ty_obj user_obj_hdl;            /* OPTIONAL */
    cpi_ty_obj wbin_obj_hdl;            /* OPTIONAL */

        /* You must pass in either case_objid or case_obj_hdl */
    cpi_ty_objid case_objid;            /* case object id */
    cpi_ty_obj case_obj_hdl;            /* case object hdl*/

                /* Subcase related information. */
    cpi_ty_obj subc_obj_hdl;            /* subcase object */
    cpi_ty_obj status_obj_hdl;          /* set by caller */
                                        /* empty by default */
    cpi_ty_obj prrty_obj_hdl;           /* set by caller */
                                        /* empty by default */
    cpi_ty_obj svrty_obj_hdl;           /* set by caller */
                                        /* empty by default */
    cpi_ty_obj state_obj_hdl;           /* set by caller */
                                        /* empty by default */
    cpi_ty_obj cascond_obj_hdl;         /* set by caller */
                                        /* empty by default */
    }
zsb_ty_new_subcase_part;

typedef struct
    {
    cpi_ty_state tstate;
    long int flags;
    void *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part;   /* PROTECTED DATA,
                                                 don't modify */
    zsb_ty_new_subcase_part new_subcase_part; /* Use this structure
                                                 */
    }
zsb_ty_subcase_struct;
```

Type Creation Functions

```
(define subcase-ptr (zsb_new_create dbid-ptr 0))
(define null-subcase-ptr (get_zsb_ty_subcase_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zsb_ty_subcase_struct? subcase-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zsb_ty_subcase_struct_int_get_options subcase-ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_user_name_max_length
subcase-ptr))
(define str-ret (zsb_ty_subcase_struct_str_get_user_name subcase-
ptr))
```

```
(define int-ret (zsb_ty_subcase_struct_int_get_user_obj_hdl subcase-
ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_wbin_obj_hdl subcase-
ptr))
(define objid-ret (zsb_ty_subcase_struct_objid_get_case_objid
subcase-ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_case_obj_hdl subcase-
ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_subc_obj_hdl subcase-
ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_status_obj_hdl
subcase-ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_prrty_obj_hdl subcase-
ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_svrty_obj_hdl subcase-
ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_state_obj_hdl subcase-
ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_cascond_obj_hdl
subcase-ptr))
(define int-ret (zsb_ty_subcase_struct_int_get_tstate subcase-ptr))
```

Field-accessing Functions (set)

```
(zsb_ty_subcase_struct_int_set_options subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_user_name_max_length subcase-ptr int-
value)
(zsb_ty_subcase_struct_str_set_user_name subcase-ptr str-value)
(zsb_ty_subcase_struct_int_set_user_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_wbin_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_objid_set_case_objid subcase-ptr objid-value)
(zsb_ty_subcase_struct_int_set_case_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_subc_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_status_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_prrty_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_svrty_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_state_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_cascond_obj_hdl subcase-ptr int-value)
(zsb_ty_subcase_struct_int_set_tstate subcase-ptr int-value)
```

zrq_ty_hdr_struct

Structure Definition

```
typedef struct
    {
    cpi_ty_obj ship_site_obj_hdl;     /* REQUIRED ship_to_site */
    cpi_ty_obj bill_site_obj_hdl;     /* OPTIONAL bill_to_site */
    cpi_ty_obj case_obj_hdl;          /* OPTIONAL; you can supply
                                             this instead of
                                             ship_site_obj_hdl */
    cpi_ty_obj country_obj_hdl;       /* OPTIONAL */
    cpi_ty_obj state_prov_obj_hdl;    /* OPTIONAL */
    cpi_ty_obj demand_hdr_obj_hdl;    /* header handle */
    }
zrq_ty_hdr_part;

typedef struct
    {
    cpi_ty_state tstate;
    long int flags;
    void *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part;/* protected data */
    zrq_ty_hdr_part demand_hdr_part;   /* use this structure */
    }
zrq_ty_hdr_struct;
```

Type Creation Functions

```
(define hdr-ptr (zrq_hdr_create dbid-ptr 0))
(define null-hdr-ptr (get_zrq_ty_hdr_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zrq_ty_hdr_struct? hdr-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zrq_ty_hdr_struct_int_get_ship_site_obj_hdl hdr-
ptr))
(define int-ret (zrq_ty_hdr_struct_int_get_bill_site_obj_hdl hdr-
ptr))
(define int-ret (zrq_ty_hdr_struct_int_get_case_obj_hdl hdr-ptr))
(define int-ret (zrq_ty_hdr_struct_int_get_country_obj_hdl hdr-ptr))
(define int-ret (zrq_ty_hdr_struct_int_get_state_prov_obj_hdl hdr-
ptr))
(define int-ret (zrq_ty_hdr_struct_int_get_demand_hdr_obj_hdl hdr-
ptr))
(define int-ret (zrq_ty_hdr_struct_int_get_tstate hdr-ptr))
```

Field-accessing Functions (set)

```
(zrq_ty_hdr_struct_int_set_ship_site_obj_hdl hdr-ptr int-value)
(zrq_ty_hdr_struct_int_set_bill_site_obj_hdl hdr-ptr int-value)
(zrq_ty_hdr_struct_int_set_case_obj_hdl hdr-ptr int-value)
(zrq_ty_hdr_struct_int_set_country_obj_hdl hdr-ptr int-value)
(zrq_ty_hdr_struct_int_set_state_prov_obj_hdl hdr-ptr int-value)
(zrq_ty_hdr_struct_int_set_demand_hdr_obj_hdl hdr-ptr int-value)
(zrq_ty_hdr_struct_int_set_tstate hdr-ptr int-value)
```

zrq_ty_dtl_struct

Structure Definition

```
typedef struct
    {
    cpi_ty_obj location_hdl;          /* OPTIONAL */
    cpi_ty_obj mod_level_obj_hdl;     /* REQUIRED */
    cpi_ty_obj part_num_obj_hdl;      /* REQUIRED */
    cpi_ty_obj site_part_obj_hdl;     /* only needed for web */
    cpi_ty_obj demand_hdr_obj_hdl;    /* REQUIRED */
    cpi_ty_obj status_obj_hdl;        /* dtl status */
    cpi_ty_obj *demand_dtl_lst;       /* array of dtl_hdls */
    cpi_ty_obj *cond_lst;             /* array of cond_hdl */
    long int count;
            /* if a part requested is a serialized item the count
                    equals number of dtl_objs. For non_serialized
                    item, there is only one dtl_obj. */
    }
zrq_ty_dtl_part;

typedef struct
    {
    cpi_ty_state tstate;
    long int flags;
    void *subclass_ptr;
    }
zpi_ty_ctxt_obj_part;

typedef struct
    {
    zpi_ty_ctxt_obj_part base_context_part;   /* protected */
    zrq_ty_dtl_part demand_dtl_part;          /* use this */
    }
zrq_ty_dtl_struct;
```

Type Creation Functions

```
(define dtl-ptr (zrq_dtl_create dbid-ptr 0))
(define null-dtl-ptr (get_zrq_ty_dtl_struct_null_ptr))
```

Type Validation Function

```
(define bool-ret (zrq_ty_dtl_struct? dtl-ptr))
```

Field-accessing Functions (get)

```
(define int-ret (zrq_ty_dtl_struct_int_get_location_hdl dtl-ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_mod_level_obj_hdl dtl-
ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_part_num_obj_hdl dtl-ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_site_part_obj_hdl dtl-
ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_demand_hdr_obj_hdl dtl-
ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_status_obj_hdl dtl-ptr))
(define ptr-ret (zrq_ty_dtl_struct_ptr_get_demand_dtl_lst dtl-ptr))
(define ptr-ret (zrq_ty_dtl_struct_ptr_get_cond_lst dtl-ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_count dtl-ptr))
(define int-ret (zrq_ty_dtl_struct_int_get_tstate dtl-ptr))
```

Field-accessing Functions (set)

```
(zrq_ty_dtl_struct_int_get_location_hdl dtl-ptr int-value)
(zrq_ty_dtl_struct_int_get_mod_level_obj_hdl dtl-ptr int-value)
(zrq_ty_dtl_struct_int_get_part_num_obj_hdl dtl-ptr int-value)
```

```
(zrq_ty_dtl_struct_int_get_site_part_obj_hdl dtl-ptr int-value)
(zrq_ty_dtl_struct_int_get_demand_hdr_obj_hdl dtl-ptr int-value)
(zrq_ty_dtl_struct_int_get_status_obj_hdl dtl-ptr int-value)
(zrq_ty_dtl_struct_ptr_get_demand_dtl_lst dtl-ptr ptr-value)
(zrq_ty_dtl_struct_ptr_get_cond_lst dtl-ptr ptr-value)
(zrq_ty_dtl_struct_int_get_count dtl-ptr int-value)
(zrq_ty_dtl_struct_int_get_tstate dtl-ptr int-value)
```

7.4 Clarify API (API Toolkit) Functions

Object Creation Routines

Object Manipulation Routines

cpi_obj_add

Description

Adds an object to the local memory bounded by a transaction state.

Syntax

```
(cpi_obj_add tstate <obj-handle> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define obj-handle (cpi_obj_make "part_number" 0))
(define bool-ret (cpi_obj_add tstate obj-handle 0))
```

cpi_obj_copy

Description

Makes a copy of an object in local memory.

Syntax

```
(cpi_obj_copy <obj-handle> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, the **object handle** for the newly copied object; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define obj-handle (cpi_obj_create tstate "part_number" 0))
(define new-obj (cpi_obj_copy obj-handle 0))
```

cpi_obj_create

Description

Creates an object in local memory and add it to a transaction state.

Syntax

```
(cpi_obj_copy <tstate> <obj-type> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
obj-type	char	Object type.
0	long int	Routine option flags.

Returns

Upon successful execution, the **object handle** for the newly created object; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define obj-handle (cpi_obj_create tstate "part_number" 0))
```

cpi_obj_make

Description

Makes an object in local memory.

Syntax

```
(cpi_obj_make <obj-type> <0>)
```

Parameters

Name	Type	Description
obj-type	char	Object type.
0	long int	Routine option flags.

Returns

Upon successful execution, the **object handle** for the new object; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-handle (cpi_obj_make "part_number" 0))
```

cpi_obj_delete

Description

Deletes an object from the database.

Syntax

```
(cpi_obj_delete <tstate> <obj-handle> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define obj-handle (cpi_obj_make "part_number" 0))
(define bool-ret (cpi_obj_add tstate obj-handle 0))
(deifne bool-ret (cpi_obj_delete tstate obj-handle 0))
```

cpi_obj_dump

Description

Dumps the contents of a local object to a file or to standard output. You can use this routine when debugging to view the contents of an object.

Syntax

```
(cpi_obj_dump <tstate> <out-file> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
out-file	FILE	Output file.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-handle (cpi_obj_make "part_number" 0))
(define null-file (get_file_ptr_null_ptr))
(define bool-ret (cpi_obj_dump obj-handle null-file 0))
```

(Given a null file, it prints output to **stdout**.)

cpi_obj_exists

Description

Checks whether an object in local memory is also in database.

Syntax

```
(cpi_obj_exists <obj-handle> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_obj_exists obj-handle 0))
```

cpi_obj_free

Description

Frees an existing local in-memory object. It does not do any deletion of objects in the database.

Syntax

```
(cpi_obj_free <obj-handle> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_obj_free obj-handle 0))
```

cpi_obj_get_field

Description

Gets the value and data type from a local object.

Syntax

```
(cpi_obj_get_field <obj-handle> <field-name> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
field-name	char	Field name.
0	long int	Routine option flags.

Returns

Upon successful execution, a **vector**; otherwise, undefined. The first component in the returned vector is the **value**, the second one is the **data type** (long int).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define res-vec (cpi_obj_get_field obj-handle "part_number" 0))
(display "Field value: ")
(display (vector-ref res-vec 0)) (newline)
(display "Data Type: ")
(display (vector-ref res-vec 1)) (newline)
```

cpi_obj_get_field_ids

Description

Gets the IDs for the specified fields.

Syntax

```
(cpi_obj_get_field_ids <obj-type> <num-names> <vec-names>)
```

Parameters

Name	Type	Description
obj-type	long int	Object type.
num-names	long int	Number names.
vec-names	vector	Vector names.

Returns

Upon successful execution, a **vector** containing other vectors; otherwise, undefined. The components of the returned vector are two-component vectors; the first component represents **id**, the second component represents **data type**.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-type (cpi_obj_get_obj_type obj-handle))
(define id-vec (cpi_obj_get_field_ids obj-type 2 '#("part_number"
"mod_level")))
(display "Part Number ID: ")
(display (vector-ref (vector-ref id-vec 0) 0)) (newline)
(display "Part Number Type: ")
(display (vector-ref (vector-ref id-vec 0) 1)) (newline)
(display "Mod Level ID: ")
(display (vector-ref (vector-ref id-vec 1) 0)) (newline)
(display "Mod Level Type: ")
(display (vector-ref (vector-ref id-vec 1) 1)) (newline)
```

cpi_obj_get_field_length

Description

Returns the maximum field length.

Syntax

```
(cpi_obj_get_field_length <obj-type> <field-name>)
```

Parameters

Name	Type	Description
obj-type	char	Object type.
field-name	char	Field name.

Returns

The maximum field length.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define length (cpi_obj_get_field_length "part_num" "part_number"))
```

cpi_obj_get_mult_fields

Description

Gets the values and data types from one or more fields from within a local object.

Syntax

```
(cpi_obj_get_mult_fields <obj-handle> <num-fields> <field-names> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
num-fields	long int	Number of fields.
field-names	vector	Field names.
0	long int	Routine option flags.

Returns

Upon successful execution, a **vector**; otherwise, undefined. The first component in the returned vector is the **value**, the second one is the **data type** (long int).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define res-vec (cpi_obj_get_mult_fields obj-handle 2
`#("part_number" "mod_level") 0))
(display "Part Number Field Value: ")
(display (vector-ref (vector-ref res-vec 0) 0) (newline)
(display "Part Number Data Type: ")
(display (vector-ref (vector-ref res-vec 0) 1)) (newline)
(display "Mod Level Field Value: ")
(display (vector-ref (vector-ref res-vec 1) 0) (newline)
(display "Mod Level Data Type: ")
(display (vector-ref (vector-ref res-vec 1) 1)) (newline)
```

cpi_obj_get_obj_type

Description

Returns the type of an object (for example, case, or contract).

Syntax

```
(cpi_obj_get_obj_type <obj-handle>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.

Returns

It returns the **object type** (long int).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-type (cpi_obj_get_obj_type obj-handle))
```

cpi_obj_id

Description

Gets the object ID number (the primary key) from an object in the database.

Syntax

```
(cpi_obj_id <obj-handle>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.

Returns

A **vector**:

- (**cpi_ty_objid** *) type Monk object
- Boolean, if the object is permanent
- object type string
- high-word portion of **objid** structure
- low-word portion of **objid** structure (**OBJID**)

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-id (cpi_obj_id obj-handle))
(display "cpi_ty_objid_ptr?: ")
(display (cpi_ty_objid_ptr? (vector-ref obj-id 0))) (newline)
(display " is object perm?: ")
(display (vector-ref obj-id 1)) (newline)
(display "object type: ")
(display (vector-ref obj-id 2)) (newline)
(display "objid high word: ")
(display (vector-ref obj-id 3)) (newline)
(display "objid low word: ")
(display (vector-ref obj-id 4)) (newline)
```

cpi_obj_num_fields

Description

Gets number of fields in the object.

Syntax

```
(cpi_obj_num_fields <obj-handle> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

The number of fields.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define num-fields (cpi_obj_num_fields obj-handle 0))
```

cpi_obj_remove

Description

Removes an object from transactional state.

Syntax

```
(cpi_obj_remove <tstate> <obj-handle> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
.
.
.
(define bool-ret (cpi_obj_remove tstate obj-handle 0))
```

cpi_obj_set_field

Description

Sets the field on an object to a specific value.

Syntax

```
(cpi_obj_set_field <obj-handle> <field-name> <field-type> <field-
value> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
field-name	char	Field name.
field-type	char	Field type.
field-value	char	Field value.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_obj_set_field obj-handle "part_number" "char
pointer" "STC-PART" 0))
```

cpi_obj_set_modified

Description

Marks an object as being modified. When the transaction state is saved, the object is either updated or inserted into the database.

Syntax

```
(cpi_obj_set_modified <obj-handle> <tstate> <modified> <0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
tstate	long int	Transaction state.
modified	long int	Modified indicator (TRUE/FALSE).
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define bool-ret (cpi_obj_set_modified obj-handle tstate 1 0))
```

cpi_obj_set_mult_fields

Description

Sets one or more fields on an object to specific values.

Syntax

```
(cpi_obj_set_mult_fields <obj-handle> <num-fields> <field-values>
<0>)
```

Parameters

Name	Type	Description
obj-handle	long int	Object handle.
num-fields	char	Number of fields.
field-values	vector	Field values.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_obj_set_mult_fields obj-handle
`#(#("part_number" "char pointer" "STC-PART")) `#(#("mod_level" "char
pointer" "STC-PART-0")) 0))
```

cpi_obj_typename

Description

Returns the object type name for the specified object type.

Syntax

```
(cpi_obj_typename <obj-type>)
```

Parameters

Name	Type	Description
obj-type	long int	Object type.

Returns

The **typename** on success; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define obj-type (cpi_obj_get_obj_type obj-handle))
(define str-type-name (cpi_obj_typename obj-type))
```

cpi_obj_update

Description

Indicates that an object in the database needs to be updated with changes made to a corresponding local object.

Syntax

```
(cpi_obj_update <tstate> <obj-handle> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
obj-handle	long int	Object handle.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define str-type-name (cpi_obj_update tstate obj-handle 0))
```

cpi_rel_fromid

Description

Relates an object in the database (identified by the object ID number) to an object in local memory.

Syntax

```
(cpi_rel_fromid <tstate> <from-objid> <to-obj-hdl> <rel-name> <is-
related> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
from-objid	cpi_ty_objid	Source object ID.
to-obj-hdl	long int	Target object handle.
rel-name	char	Name of relation.
is-related	long int	Related indicator (TRUE/FALSE).
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define str-type-name (cpi_rel_fromid tstate from-objid to-obj-handle
"part_num2part_class" 1 0))
```

cpi_rel_ids

Description

Relates two objects in the database, identified by their respective object ID numbers.

Syntax

```
(cpi_rel_ids <tstate> <from-objid> <to-objid> <rel-name> <is-related>
<0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
from-objid	cpi_ty_objid	Source object ID.
to-objid	cpi_ty_objid	Target object handle.
rel-name	char	Name of relation.
is-related	long int	Related indicator (TRUE/FALSE).
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define str-type-name (cpi_rel_fromid tstate from-objid to-objid
"part_num2part_class" 1 0))
```

cpi_rel_objs

Description

Relates two local objects.

Syntax

```
(cpi_rel_objs <tstate> <from-obj-hdl> <to-obj-hdl> <rel-name> <is-
related> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
from-obj-hdl	long int	Source object ID.
to-obj-hdl	long int	Target object handle.
rel-name	char	Name of relation.
is-related	long int	Related indicator (TRUE/FALSE).
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define str-type-name (cpi_rel_objs tstate from-obj-hdl to-obj-hdl
"part_num2part_class" 1 0))
```

cpi_rel_toid

Description

Relates an object in local memory to an object in the database, identified by the object ID number.

Syntax

```
(cpi_rel_toid <tstate> <from-obj-hdl> <to-objid> <rel-name> <is-
related> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
from-obj-hdl	long int	Source object ID.
to-objid	cpi_ty_objid	Target object handle.
rel-name	char	Name of relation.
is-related	long int	Related indicator (TRUE/FALSE).
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
(define str-type-name (cpi_rel_toid <tstate> <from-obj-hdl> <to-
objid> "part_num2part_class" 1 <0>))
```

cpi_qry_free_results

Description

Frees memory used by the traversal map structure, which stores the results of the **cpi_qry_traverse_map** routine.

Syntax

```
(cpi_qry_free_results <rescol>)
```

Parameters

Name	Type	Description
rescol	cpi_ty_rescol	Column of results.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define fcond1 (get_cpi_ty_fcond_ptr '#(#(4 "part_number" "CDS"))))
(define condition (get_cpi_ty_cond_ptr 1 fcond1 1))
(define forder1 (get_cpi_ty_forder_ptr "part_number" 1))
(define forder2 (get_cpi_ty_forder_ptr "model_num" 2))
(define order (get_cpi_ty_order_ptr 2 (list->vector (list forder1
forder2))))
(define trav_map1 (get_cpi_ty_trav_map_ptr "" "part_num" 0 4
condition order 1))
(define rescol-ptr (cpi_qry_traverse_map <dbid-ptr> 0 <null-objid>
(list->vector (list trav_map1)) 0))
(define bool-ret (cpi_qry_free_results <rescol-ptr>))
```

cpi_obj_get_result

Description

Gets the results of one query from the traversal map structure, which stores the results of the cpi_qry_traverse_map routine.

Syntax

```
(cpi_obj_get_result <rescol> <trav-index> <0>)
```

Parameters

Name	Type	Description
rescol	cpi_ty_rescol	Column of results.
trav-index	long int	Traverse map index.
0	long int	Routine option flags.

Returns

Upon success, a **vector** of object handles for resulted objects; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define fcond1 (get_cpi_ty_fcond_ptr `#(#(4 "part_number" "CDS"))))
(define condition (get_cpi_ty_cond_ptr 1 fcond1 1))
(define forder1 (get_cpi_ty_forder_ptr "part_number" 1))
(define forder2 (get_cpi_ty_forder_ptr "model_num" 2))
(define order (get_cpi_ty_order_ptr 2 (list->vector (list forder1
forder2))))
(define trav_map1 (get_cpi_ty_trav_map_ptr "" "part_num" 0 4
condition order 1))
(define rescol-ptr (cpi_qry_traverse_map dbid-ptr 0 null-objid (list-
>vector (list trav_map1)) 0))
(define res-vec (cpi_qry_get_result rescol-ptr 0 0))
```

cpi_obj_get_result

Description

Runs the queries described by traversal map structure.

Syntax

```
(cpi_obj_get_result <dbid-ptr> <tstate> <root-objid> <trav-map> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
tstate	long int	Transaction state.
root-objid	cpi_ty_objid	Object ID of starting object.
trav-map	vector	Traversal map array.
0	long int	Routine option flags.

Returns

Upon success, the (**cpi_ty_rescol** *) custom Monk object; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define fcond1 (get_cpi_ty_fcond_ptr `#(#(4 "part_number" "CDS"))))
(define condition (get_cpi_ty_cond_ptr 1 fcond1 1))
(define forder1 (get_cpi_ty_forder_ptr "part_number" 1))
(define forder2 (get_cpi_ty_forder_ptr "model_num" 2))
(define order (get_cpi_ty_order_ptr 2 (list->vector (list forder1
forder2))))
(define trav_map1 (get_cpi_ty_trav_map_ptr "" "part_num" 0 4
condition order 1))
(define null-objid (get_cpi_ty_objid_null_ptr))
(define rescol-ptr (cpi_qry_traverse_map dbid-ptr 0 null-objid (list-
>vector (list trav_map1)) 0))
```

cpi_qry_trav_map_init

Description

Initializes the query traversal map structure.

Syntax

```
(cpi_qry_trav_map_init <traverse_map> <0>)
```

Parameters

Name	Type	Description
traverse_map	vector	Query traverse map list.
0	long int	Routine option flags.

Returns

Upon success, returns an integer of 1; otherwise, 0 if failed.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Example

```
(define trav_map1 (get_cpi_ty_trav_map_ptr "" "part_num" 0 4 cond1
order1 1))
(cpi_qry_trav_map_init (list->vector (list trav_map1)) 30)
```

cpi_sess_get_option

Description

Gets the current settings for the database session.

Syntax

```
(cpi_sess_get_option <dbid-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
0	long int	Routine option flags.

Returns

Upon success, a **vector** containing four components; otherwise, undefined. The returned vector contains the following:

1. long option
2. short option
3. bool option
4. string option

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define ret-vec (cpi_sess_get_option dbid-ptr 0))
```

cpi_sess_login

Description

Logs into the Clarify database and establishes an active session.

Syntax

```
(cpi_sess_login <server-name> <database-name> <login-name>
<password>)
```

Parameters

Name	Type	Description
server-name	char	Server name.
database-name	char	Database name.
login-name	char	User's login name.
password	char	User's login password.

Returns

Upon success, a (**cpi_ty_dbid** *) custom Monk object; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
```

cpi_sess_logout

Description

Logs off the Clarify database and ends the active session.

Syntax

```
(cpi_sess_logout <dbid-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
.
.
.
(define bool-ret (cpi_sess_logout dbid-ptr 0))
```

cpi_sess_disconnect

Description

Disconnects you from the Clarify database that you are logged into.

Syntax

```
(cpi_sess_disconnect)
```

Parameters

None

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define db-handle (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))

(if db-handle
(display "GOOD\n")
(display "BAD\n")
)
(cpi_sess_disconnect)
(cpi_sess-reconnect)

(cpi_sess_logout)
```

cpi_sess_reconnect

Description

Reconnects you to the Clarify database that you were logged into and subsequently disconnected from.

Syntax

```
(cpi_sess_reconnect)
```

Parameters

None

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define db-handle (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))

(if db-handle
(display "GOOD\n")
(display "BAD\n")
)
(cpi_sess_disconnect)
(cpi_sess-reconnect)

(cpi_sess_logout)
```

cpi_sess_ping

Description

Checks to see if the server connection is still valid.

Syntax

```
(cpi_sess_ping <dbid-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, a Boolean false (**#f**).

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
.
.
.
(define bool-ret (cpi_sess_ping dbid-ptr 0))
```

cpi_sess_set_option

Description

Sets the database settings.

Syntax

```
(cpi_sess_set_option <dbid-ptr> <0> <long-option> <short-option>
<bool-option> <string-option>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
0	long int	Routine option flags.
long-option	long int	Long option.
short-option	long int	Short option.
bool-option	long int	Boolean option.
string-option	char	String option.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define bool-ret (cpi_sess_set_option dbid_ptr 2 0 0 1 ""))
```

cpi_state_create

Description

Creates a new transactional state in local memory.

Syntax

```
(cpi_state_create <dbid-ptr> <0>)
```

Parameters

Name	Type	Description
dbid-ptr	cpi_ty_dbid	Database ID pointer.
0	long int	Routine option flags.

Returns

Upon success, the **transactional state**; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
```

cpi_state_commit

Description

Commits a transaction state to the database, updating objects and relations in a single transaction. If this single transaction fails, all changes to the database are rolled back.

Syntax

```
(cpi_state_commit <tstate> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
.
.
.
(define bool-ret (cpi_state_commit tstate 0))
```

cpi_state_dump

Syntax

```
(cpi_state_dump <tstate> <out-file> <0>)
```

Description

Dumps the contents of a transaction state to a file or to standard output. When you are debugging, you can use this routine to view the contents of the transaction state.

Parameters

Name	Type	Description
tstate	long int	Transaction state.
out-file	FILE	Output file.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
.
.
.
(define dump-file (get_fileptr_ptr "c:/temp/dump.log"))
(define bool-ret (cpi_state_dump tstate dump-file 0))
```

cpi_state_rollback

Description

Rolls back and deletes a transaction state. You can use this routine to free the memory used by a transaction state.

Syntax

```
(cpi_state_rollback <tstate> <0>)
```

Parameters

Name	Type	Description
tstate	long int	Transaction state.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define tstate (cpi_state_create dbid-ptr 0))
.
.
.
(define bool-ret (cpi_state_commit tstate 0))
(define bool-ret (cpi_state_rollback tstate 0))
```

cpi_alert_clear

Description

Clears any alerts in current alert stack.

Syntax

```
(cpi_alert_clear)
```

Parameters

None.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_alert_clear))
```

cpi_alert_print

Description

Prints the current alert stack onto a file.

Syntax

```
(cpi_alert_print <log-file> <0>)
```

Parameters

Name	Type	Description
log-file	char	Log file name.
0	long int	Routine option flags.

Returns

Upon successful execution, a Boolean true (**#t**); otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_alert_print "C:\temp\debug.log" 0))
```

cpi_utl_num_gen

Description

Generates an identification number based on numbering schemes set up in the Clarify system.

Syntax

```
(cpi_utl_num_gen <schema-name> <max-length>)
```

Parameters

Name	Type	Description
schema-name	char	Schema name.
max-length	long int	Maximum length of ID number.

Returns

Returns identification string on success; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define bool-ret (cpi_utl_num_gen "part_num" 24))
```

cpi_utl_current_time

Description

Retrieves the current time.

Syntax

```
(cpi_utl_current_time)
```

Parameters

None.

Returns

Upon success, the **time**; otherwise, undefined.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(define current-time (cpi_utl_current_time))
```

To retrieve the current time:

```
cpi_ty_status cpi_utl_current_time (string) char *string;
```

7.5 Clarify API (API Toolkit) Structures

cpi_ty_fcond

Structure Definition

```
typedef struct
    {
    cpi_ty_op op;
    char *fname;
    char *value;
    }
cpi_ty_fcond;
```

Type Creation Functions

```
(define fcond-ptr (get_cpi_ty_fcond_ptr `#(#(CPI_OP_EQUAL
"part_number" "CDS"))))
(define null-fcond (get_cpi_ty_fcond_null_ptr))
```

Type Validation Function

```
(define bool-ret (cpi_ty_fcond_ptr? fcond-ptr))
```

cpi_ty_cond

Structure Definition

```
typedef struct
    {
    long int num_conds;
    cpi_ty_fcond *conds;
    cpi_ty_bool and;
    }
cpi_ty_cond;
```

Type Creation Functions

```
(define cond-ptr (get_cpi_ty_cond_ptr 2 ;; number of filters fcond-
   ptr ;; (cpi_ty_fcond *) object 1 ;; and))
(define null-cond (get_cpi_ty_cond_null_ptr))
```

Type Validation Function

```
(define bool-ret (cpi_ty_cond_ ptr? cond-ptr))
```

cpi_ty_forder

Structure Definition

```
typedef struct
{
char *fname;
cpi_ty_bool ascending;
}
cpi_ty_forder;
```

Type Creation Functions

```
(define forder-ptr (get_cpi_ty_forder_ptr "part_number" 1))
(define null-forder (get_cpi_ty_forder_null_ptr))
```

Type Validation Function

```
(define bool-ret (cpi_ty_forder_ptr? forder-ptr))
```

cpi_ty_order

Structure Definition

```
typedef struct
{
long int num_orders;
cpi_ty_forder *orders;
}
cpi_ty_order;
```

Type Creation Functions

```
(define cpi_ty_order-ptr (get_cpi_ty_order_ptr 124 (vector forder1-
ptr forder2-ptr)))
;forder1-ptr and forder2-ptr is of cpi_ty_forder type
```

Type Validation Function

```
(define bool-ret (cpi_ty_order_ptr? cpi_ty_order-ptr))
```

cpi_ty_objid

Structure Definition

```
typedef struct
    {
    long int high_word;
    cpi_ty_lowid low_word;
    long int sub_id;
    }
cpi_ty_objid;
```

Type Creation Functions

```
(define objid-ptr (get_cpi_ty_objid_ptr 124 ;; high word 26843556 ;;
low word 0 ;; sub ID))
(define null-objid (get_cpi_ty_objid_null_ptr))
```

Type Validation Function

```
(define bool-ret (cpi_ty_objid_ptr? objid-ptr))
```

cpi_ty_trav_map

Structure Definition

```
typedef struct
    {
    char *rel_name;       /* Used only for 'traversal' queries */
    char *object_type;    /* Used only for 'flat' queries */
    long int parent_ind; /* Used for hierarchical queries */
    long int options;     /* See CPI_QRY_TRAV options */
    cpi_ty_cond conds;    /* Conditions on 'result objects' */
    cpi_ty_order orders; /* Sorting order for results of query */
    cpi_ty_result results_handle;
                          /* Results; filled by query routine */
    }
cpi_ty_trav_map;
```

Type Creation Functions

```
(define trav-map-ptr (get_cpi_ty_trav_map_ptr "" ;; relation name
"part_num" ;; table name 0 ;; parent traverse map index 4 ;; Options
cond-ptr ;; condition order-ptr ;; sorting order 1 ;; unused))
(define null-trav-map (get_cpi_ty_trav_map_null_ptr))
```

Type Validation Function

```
(define bool-ret (cpi_ty_trav_map_ptr? trav-map-ptr))
```

cpi_ty_dbid

Structure Definition

```
typedef long int cpi_ty_dbid;
```

Type Creation Functions

```
(define dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
(define null-dbid (get_cpi_ty_dbid_null_ptr))
```

Type Validation Function

```
(define bool-ret (cpi_ty_dbid_ptr? dbid-ptr))
```

7.6 Miscellaneous Functions and Structures

read_alert

Description

Reads alerts from a file.

Syntax

```
(read_alert <file-path>)
```

Parameters

Name	Type	Description
file-path	char *	Path to specified file.

Return Values

Upon successful execution, the function returns a string format of alerts read from the file specified by the **file-path**.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Example

```
(define err-message (read_alert "C:\TEMP\debug.log"))
fileptr
structure definition
typedef FILE *fileptr;
```

Type Create Functions

```
(define file-ptr (get_fileptr_ptr "C:\TEMP\tmpfile"))
(define null-file (get_fileptr_null_ptr))
```

Type Validation Functions

```
(define bool-ret (cpi_ty_fileptr_ptr? file-ptr))
```

7.7 Clarify Monk Functions

These Monk functions have been developed specifically to facilitate working with the Clarify application, specifically dealing with basic database-accessing tasks and error reporting. These functions can be categorizes as:

Connectivity Functions on page 176

Database Accessing Functions on page 180

Error-handling Functions on page 190

7.7.1 Connectivity Functions

clarify-startup on page 177

clarify-connection-retry on page 178

clarify-finish on page 179

clarify-startup

Description

The default e*Way startup function.

Syntax

```
(clarify-startup)
```

Parameters

None.

Return Values

None.

Throws

None.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

clarify-connection-retry

Description

The default function for requesting that the e*Way attempt to reconnect to the Clarify application.

Syntax

```
(clarify-connection-retry)
```

Parameters

None.

Return Values

None.

Throws

None.

Location

clarify-connection-retry.monk

clarify-finish

Description

The default e*Way shutdown function.

Syntax

```
(clarify-finish)
```

Parameters

None.

Return Values

None.

Throws

None.

Location

clarify-finish.monk

7.7.2 Database Accessing Functions

clarify-lookup

Description

Conducts a query using the **OBJID** field found on the passed table node Monk structure. The resulting field values are filled onto the same Monk structure.

Syntax

```
(clarify-lookup <table-node> <tstate>)
```

Parameters

Name	Type	Description
table-node	node path	Table node containing desired OBJID field.
tstate	long int	Transactional state of table identified by <table-node>.

Returns

Upon successful execution, a **vector** of found object handles or an **empty vector** if no match was found. Upon failure, a Boolean false (**#f**), and the error code **clarify_error** is set.

Throws

None.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(let
    (
        (output (make-message-structure
TABLE_PART_DOMAIN-delm
TABLE_PART_DOMAIN-struct)
        )
        (dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
        (tstate (cpi_state_create dbid-ptr 0))
    )
    (define look_res (clarify-struct-lookup
~output%TABLE_PART_DOMAIN.TABLE_PRT_DOMAIN
tstate)
    )
)
```

Additional Information

Regarding the transactional state, refer to the following functions:

- **cpi_state_create** on page 160
- **cpi_state_commit** on page 161
- **cpi_state_dump** on page 162
- **cpi_state_rollback** on page 163.

clarify-insert

Description

First creates a new object, using the values, with flags assigned as either **I** or **B**, found on the **table-node** Monk structure, then insert the new object into the database.

Syntax

```
(clarify-insert <table-node> <tstate>)
```

Parameters

Name	Type	Description
table-node	node path	Table node containing desired OBJID field.
tstate	long int	Transactional state of table identified by <table-node>.

Returns

Upon successful execution, a **vector** of found object handles or an **empty vector** if no match was found. Upon failure, a Boolean false (**#f**), and the error code **clarify_error** is set.

Throws

None.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(let
    (
        (input (make-message-structure
TABLE_PART_DOMAIN-delm
TABLE_PART_DOMAIN-struct)
        )
        (dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
        (tstate (cpi_state_create dbid-ptr 0))
    )
    (define obj-handle (clarify-struct-insert
~input%TABLE_PART_DOMAIN.TABLE_PRT_DOMAIN
tstate)
    )
)
```

Additional Information

Regarding the transactional state, refer to the following functions:

- **cpi_state_create** on page 160
- **cpi_state_commit** on page 161
- **cpi_state_dump** on page 162
- **cpi_state_rollback** on page 163.

clarify-update

Description

First does a query using the **OBJID** value found on the **table-node** Monk structure, then performs an update operation for the valid values, with flags assigned as either **U** or **B**, found on the same structure.

Syntax

```
(clarify-update <table-node> <tstate>)
```

Parameters

Name	Type	Description
table-node	node path	Table node containing desired OBJID field.
tstate	long int	Transactional state of table identified by <table-node>.

Returns

Upon successful execution, a **vector** of found object handles or an **empty vector** if no match was found. Upon failure, a Boolean false (**#f**), and the error code **clarify_error** is set.

Throws

None.

Location

Clarify 8 Windows and UNIX

stc_monkclarify.dll

Clarify 10.1 Windows

stc_monkclarify_10.dll

Clarify 10.1 UNIX

stcewclarifymonk.exe

Examples

```
(let
    (
        (input (make-message-structure
TABLE_PART_DOMAIN-delm
TABLE_PART_DOMAIN-struct)
        )
        (dbid-ptr (cpi_sess_login "CLFY" "CLFY" "USER" "Passwd"))
        (tstate (cpi_state_create dbid-ptr 0))
    )
    (define update-res (clarify-struct-update
~input%TABLE_PART_DOMAIN.TABLE_PRT_DOMAIN
tstate)
    )
)
```

Additional Information

Regarding the transactional state, refer to the following functions:

- **cpi_state_create** on page 160
- **cpi_state_commit** on page 161
- **cpi_state_dump** on page 162
- **cpi_state_rollback** on page 163.

clarify-simple-query

Description

Performs a flat query to the table **table-name**, based on filter conditions specified in **query-fcond** AND/OR the transactional state **tstate**.

Syntax

```
(clarify-simple-query <table-name> <query-fcond> <log-op> <tstate>)
```

Parameters

Name	Type	Description
table-name	string	Name of table to be queried.
query-fcond	vector	Query filter conditions.
log-op	long int	Logical operator (CLF_COND_AND or CLF_COND_OR).
tstate	long int	Transactional state of table identified by <table-name>.

Returns

Upon successful execution, a **vector** of found object handles or an **empty vector** if no match was found. Upon failure, a Boolean false (**#f**), and the error code **clarify_error** is set.

Throws

None.

Location

clarify-simple-query.monk

Examples

```
(define query-results (clarify-simple-query "part_num";; table-name
`#(;; filter cond. #(CPI_OP_EQUAL "part_number" "CDS")) CLF_COND_AND
;; AND tstate    ;; trans. state))
```

Additional Information

Regarding the transactional state, refer to the following functions:

- **cpi_state_create** on page 160
- **cpi_state_commit** on page 161
- **cpi_state_dump** on page 162
- **cpi_state_rollback** on page 163.

clarify-complex-query

Description

Performs a query to the table **table-name** in the transactional state **tstate**.

Syntax

```
(clarify-complex-query <table-name> <tstate>)
```

Parameters

Name	Type	Description
table-name	string	Name of table to be queried.
tstate	long int	Transactional state of table identified by <table-name>.

Returns

Upon successful execution, a **vector** of found object handles or an **empty vector** if no match was found. Upon failure, a Boolean false (**#f**) and the error code **clarify_error** is set.

Throws

None.

Location

clarify-complex-query.monk

Examples

```
(define query-results (clarify-complex-query tstate ;; trans. state))
```

Additional Information

Regarding the transactional state, refer to the following functions:

- **cpi_state_create** on page 160
- **cpi_state_commit** on page 161
- **cpi_state_dump** on page 162
- **cpi_state_rollback** on page 163.

clarify-relation-query

Description

Returns one or many related objects in the **tstate** transactional state, given the object handle **obj-handle** and relation name **relation-name**.

Syntax

```
(clarify-relation-query <obj-handle> <relation-name> <tstate>)
```

Parameters

Name	Type	Description
obj-handle	long int	Specified object handle.
relation-name	string	Specified relation name.
tstate	long int	Transactional state of table identified by <table-name>.

Returns

Upon successful execution, a **vector** of found object handles or an **empty vector** if no match was found. Upon failure, a Boolean false (**#f**), and the error code **clarify_error** is set.

Throws

None.

Location

clarify-relation-query.monk

Examples

```
(define query-results (clarify-relation-query obj-handle ;;
    object"address2country";; rel. name Tstate;; trans. state))
```

Additional Information

Regarding the transactional state, refer to the following functions:

- **cpi_state_create** on page 160
- **cpi_state_commit** on page 161
- **cpi_state_dump** on page 162
- **cpi_state_rollback** on page 163.

7.7.3 Error-handling Functions

clarify-get-error

Syntax

```
(clarify-get-error <log-file>)
```

Description

Reads Clarify alerts and/or e*Way error message from the **log-file**, then converts them into string format for subsequent processing.

Parameters

Name	Type	Description
log-file	pathname	Name of the log file.

Returns

Error messages containing either Clarify API error or e*Way error.

Throws

None.

Location

clarify-get-error.monk

Examples

```
(define err-msg (clarify-get-error cpi_debug_file))
```

clarify-clear-error

Description

Clears both the Clarify API's alert stack and the e*Way's error messages.

Syntax

```
(clarify-clear-error)
```

Parameters

None.

Returns

None.

Throws

None.

Location

clarify-clear-error.monk

Examples

```
(dgw-clarify-clear-error)
```

clarify-error-process

Description

Retrieves errors from the log file **cpi-debug-file** using **clarify-get-error**, logs the error in an new error log file, then clears the error using **clarify-clear-error** and reconnects to the Clarify application using **clarify-connection-retry**.

Syntax

```
(clarify-error-process)
```

Parameters

None.

Returns

None.

Throws

None.

Location

clarify-error-process.monk

7.8 Basic Functions

These functions are implemented in the e*Way Kernel layer and control the e*Way's most basic operations. They are located in **stcewgenericmonk.exe**, and consist of:

start-schedule

Description

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

Syntax

```
(start-schedule)
```

Parameters

None.

Returns

None.

Throws

None.

stop-schedule

Description

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

Syntax

```
(stop-schedule)
```

Parameters

None.

Returns

None.

Throws

None.

send-external-up

Description

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

Syntax

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

Parameters

None.

Returns

None.

Throws

None.

send-external-down

Description

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

Syntax

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

Parameters

None.

Returns

None.

Throws

None.

get-logical-name

Description

Returns the logical name of the e*Way.

Syntax

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

Parameters

None.

Returns

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

Throws

None.

event-send-to-egate

Description

Sends data that the e*Way has already received from the external system into the e*Gate system as an Event.

Syntax

```
(event-send-to-egate <string>)
```

Parameters

Name	Type	Description
string	string	The data to be sent to the e*Gate system

Returns

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

Throws

None.

Additional information

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

event-send-to-egate-no-commit

Description

Sends data that the e*Way has received from the external system on to the e*Gate system as an Event—but without Committing, pending confirmation from the external system of correct transmission of the data.

Syntax

```
(event-send-to-egate-no-commit <string>)
```

Parameters

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

Returns

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

Throws

None.

event-commit-to-egate

Description

Commits the Event sent previously to the e*Gate system using **event-send-to-egate-no-commit**.

Syntax

```
(event-commit-to-egate <string>)
```

Parameters

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

Returns

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

Throws

None.

event-rollback-to-egate

Description

Rolls back the Event sent previously to the e*Gate system using **event-send-to-egate-no-commit**, following receipt of a rollback command from the external system.

Syntax

```
(event-rollback-to-egate <string>)
```

Parameters

Name	Type	Description
string	string	The data to be rolled back to the e*Gate system.

Returns

Boolean true (**#t**) if the data is rolled back successfully; otherwise, false (**#f**).

Throws

None.

insert-exchange-data-event

Description

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

Syntax

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

Parameters

None.

Returns

None.

Throws

None.

See also

Exchange Data Interval on page 54

Zero Wait Between Successful Exchanges on page 55

shutdown-request

Description

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

Syntax

```
(shutdown-request)
```

Parameters

None.

Returns

None.

Throws

None.

Additional Information

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

Index

A

C

D

E

F

G

I

J

L

M

N

P

R

S

U

Z