e*Way Intelligent Adapter for Sybase User's Guide

Release 4.5.3

Monk Version

SEEBEYOND™

Contents

Chapter 4

Implementation 35

Chapter 5

Sybase e*Way Functions 82

Introduction

SeeBeyond™ developed the e*Way Intelligent Adapter for Sybase as a graphically-configurable e*Way. The Sybase e*Way implements the logic that sends Events (data) to e*Gate and queues the next Event for processing and transport to the database.

A Monk database access library is available to log into the database, issue Structured Query Language (SQL) statements, and call stored procedures. The Sybase e*Way uses Monk to execute user-supplied database access Monk scripts to retrieve information from or send information to a database. The fetched data (information) can be returned in a Monk Collaboration which simplifies the accessibility of each column in the database table. This document describes how to install and configure the Sybase e*Way.

Note: *For information on installation, configuration, and implementation of the Java-enabled e*Way Intelligent Adapter for Sybase, see the* **e*Way Intelligent Adapter for Sybase User's Guide (Java-enabled)—Sybase_eWay_Java.pdf**.

1.1 Overview

The e*Way Intelligent Adapter for Sybase provides for data exchange between the e*Gate system and one or more Sybase databases.

The e*Gate System consists of a set of e*Ways, BOBs and Intelligent Queues, interacting with each other and configured to meet the requirements of a business application.

The Sybase e*Way can be one of many e*Ways that comprise the e*Gate System. Using one or more Sybase e*Ways, e*Gate can link Sybase-based applications with other software applications.

1.1.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system.

The reader should have expert-level knowledge of Windows NT and/or UNIX operations and administration, be thoroughly familiar with Sybase and SQL and be thoroughly familiar with Windows-style GUI operations.

1.1.2 Sybase e*Way Components

The Sybase e*Way is comprised of the following:

- **stcewgenericmonk.exe**, the executable component.
- Configuration file, which the e*Way Editor uses to define configuration parameters.
- Monk external function scripts.
- Monk extension library for Sybase database.
- e*Way Monk functions.

A complete list of installed files appears in **Figure 1 on page 12** and **Table 1 on page 13**.

Monk Extensions

A Monk extension library provides for SQL calls. The Monk library contains functions which initialize connections ("login") to a database. Other Monk functions process SQL statements or return data from the database to the Sybase e*Way.

The Monk functions supporting Sybase are not part of the basic Monk system. To obtain this functionality a Monk extension library, provided as part of the Sybase e*Way product, is automatically loaded when the Sybase e*Way is initialized.

The system developer should understand and be able to write valid Monk code.

1.2 System Requirements

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

- Windows XP
- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3 and 5.1
- HP-UX 11.0 and HP-UX 11i
- Compaq Tru64 V4.0F, V5.0A, and V5.1A
- Japanese Windows XP
- Japanese Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Japanese Windows NT 4.0 SP6a
- Japanese HP-UX 11.0

Note: *HP-UX 11.0, HP-UX 11i, and AIX 4.3.3 and 5.1 do not support Sybase database 12.5 and Sybase Open Client 12.5.*

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

- An e*Gate Participating Host, version 4.5.1 or later. For Windows XP operating system, you need an e*Gate Participating Host, version 4.5.3.

- A TCP/IP network connection.

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

1.2.1 External System Requirements

To enable the e*Way to communicate properly with the external system, the following are required:

- An external Sybase database (version 11.9, 12, or 12.5).

- A Sybase Open Client (version 11.1.1, 12, or 12.5).

Note: *For Compaq Tru64, Sybase Open Client version 12.0, or 12.5 is required.*

- Although the Sybase e*Way can function on a Registry Host on a UNIX platform, the e*Gate Enterprise Manager must be installed on a Windows NT or Windows 2000 system.

Installation

The installation procedure depends upon the operating system of the Participating Host on which you are installing the Sybase e*Way. This chapter discusses the decisions you must make prior to installation, and the procedures for installing the Sybase e*Way on either Windows or UNIX. This chapter assumes that the Sybase Open Client is installed and configured. For details about installing the Sybase Open Client for Windows or UNIX, refer to the respective installation guides from Sybase.

2.1 Installation Decisions

This section presents decisions to be made before beginning the installation. These decisions apply to both UNIX and Windows installations:

- Determine the operating system/platform on which the Sybase e*Way will operate.

- Ensure that the required version of Sybase Open Client is installed on your system.

Note: *Sybase Open Client must also be installed on a Windows unit that is running the e*Gate Enterprise Manager for an e*Gate Registry Host on a UNIX platform.*

On UNIX:

1 Issue the command below to determine the version of Sybase Open Client installed:

```
isql -v
```

2 The following output shows that version 11.1 is installed.

```
Sybase CTISQL Utility/11.1/GA/sun_svr4/SPARC Solaris 2.4/1/EBF
7353/OPT/Fri Jun  6 11:09:18 1997
```

1 Or, issue these commands:

```
cd $SYBASE/lib
strings libct.a|grep Sybase
```

2 The following output shows that version 11.1 is installed.

```
Sybase Client-Library/11.1/GA/sun_svr4/SPARC Solaris 2.4/1/EBF
7353/OPT/Fri Jun  6 11:09:18 1997
```

On Windows:

1 Issue the command below to determine the version of Sybase Open Client installed:

```
cd %SYBASE%\bin
dir
```

2.2 Installing the Sybase e*Way on Windows NT and Windows 2000

2.2.1 Pre-installation

1 Exit all Windows programs before running the setup program, including any anti-virus applications.

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

2.2.2 Installation Procedure

To install the Sybase e*Way on a Windows NT or a Windows 2000 system:

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

2 Insert the e*Way installation CD-ROM into the CD-ROM drive.

3 If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows NT Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

4 The InstallShield setup application will launch. Follow the on-screen instructions to install the e*Way.

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

2.3 Installation Directories and Files on Windows

The Sybase e*Way CD-ROM contains the following files, which the InstallShield Wizard copies to the indicated directories on your computer, creating them if necessary.

These files are installed in the Registry during your initial installation. The first time you access the e*Way to configure it, the following files (with the exception of all the Monk files) move to the Client directory.

Figure 1   Installation Directories and Files on Windows

Install Directory	Files
bin\	stcstruct.exe stc_dbapps.dll stc_dbmonkext.dll stc_dbsy11.dll stc_dbsy12.dll
configs\stcewgenericmonk\	dart.def
monk_library	dart.gui
monk_library\dart\	db-struct-bulk-insert.monk db-struct-call.monk db-struct-execute.monk db-struct-fetch.monk db-struct-insert.monk db-struct-select.monk db-struct-update.monk db-stdver-eway-funcs.monk db_bind.monk db-stdver-eway-funcs.monk db-sanitize-symbol.monk db_bind.monk db2msg-display.monk db2msg.ssc sybmsg-display.monk sybmsg.ssc oramsg-display.monk oramsg.ssc odbcmsg-display.monk odbcmsg.ssc

2.4   Installing the Sybase e*Way on UNIX

2.4.1 Pre-installation

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

2.4.2 Installation Procedure

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

1   Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.

2   If necessary, mount the CD-ROM drive.

3　At the shell prompt, type

cd /cdrom

4　Start the installation script by typing:

setup.sh

5　A menu of options will appear. Select the "install e*Way" option. Then, follow 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.***

The CD-ROM contains the following files, which are copied to the indicated path on your computer. The files and directories are under **EGate/Server/registry/repository/default/**

Table 1　Installation Directories and Files (UNIX)

Install Directory	Files
bin/	stc_dbapps.dll stc_dbmonkext.dll stc_dbsy11.dll stc_dbsyb12.dll stcstruct.exe
configs/stcewgenericmonk/	dart.def
monk_library	dart.gui
monk_library/dart/	db-struct-bulk-insert.monk db-struct-call.monk db-struct-execute.monk db-struct-fetch.monk db-struct-insert.monk db-struct-select.monk db-struct-update.monk db-stdver-eway-funcs.monk db_bind.monk db-stdver-eway-funcs.monk db-sanitize-symbol.monk db_bind.monk db2msg-display.monk db2msg.ssc sybmsg-display.monk sybmsg.ssc oramsg-display.monk oramsg.ssc odbcmsg-display.monk odbcmsg.ssc

Configuration

Before you can run the Sybase e*Way, you must configure it using the e*Way Editor, which is accessed from the e*Gate Enterprise Manager GUI. The Sybase e*Way package includes a default configuration file which you can modify using this window.

This chapter describes the procedure for configuring a new e*Way. You can also edit an existing e*Way and rename an e*Way. Procedures for creating and editing e*Gate components are provided in the Enterprise Manager's online help.

3.1 e*Way Configuration Parameters

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

To change e*Way configuration parameters:

1 In the 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.

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

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

- General Settings
- Communication Setup
- Monk Configuration
- Database Setup

3.1.1 General Settings

The General Settings control basic operational parameters.

Journal File Name

Description

Specifies the name of the journal file, which stores messages that are not picked up from the queue.

Required Values

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

Additional Information

The Journal File is used for the following conditions:

- Journal a message when it exceeds the number of retries.

- When its receipt is due to an external error, but Forward External Errors is set to **No**. (See **"Forward External Errors" on page 15** for more information.)

Max Resends Per Message

Description

Specifies the maximum number of times the e*Way attempts to resend a message to the external system after receiving an error. When this maximum number 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 the e*Way allows. When the specified number of failed messages is reached and journaled, the e*Way shut downs 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** and are received from the external system, are queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 29** for more information.

Required Values

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

3.1.2 Communication Setup

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

Note: *The schedule that you set using the e*Way's properties in the Enterprise Manager controls when the e*Way executable runs. The schedule that you set within the e*Way Editor determines when data is exchanged. Be sure that you set the "exchange data" schedule to fall within the "run the executable" schedule.*

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

- One or more specific dates/times

- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Since months do not all contain equal numbers of days, be sure not to provide boundaries that cause an invalid date selection (i.e. the 30th of every month does not include February).

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

- **Exchange Data with External Function** on page 29

- **Positive Acknowledgment Function** on page 31

- **Negative Acknowledgment Function** on page 32

If you do not define these parameters, 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 and Negative Acknowledgment functions), and whether the connection to the external system is active. If no ACK/NAK is pending and the connection is active, the e*Way immediately executes the **Exchange Data with External** function. Thereafter, the **Exchange Data with External** function is called according to the **Exchange Data Interval** parameter until the **Stop Exchange Data Schedule** time is reached.

See **"Exchange Data with External Function" on page 29**, **"Exchange Data Interval" on page 17**, and **"Stop Exchange Data Schedule" on page 17** for more information.

Stop Exchange Data Schedule

Description

Establishes the schedule to stop data exchange.

Required Values

One of the following:

- One or more specific dates/times

- A single repeating interval (such as yearly, weekly, monthly, daily, or every *n* seconds).

Since months do not all contain equal number of days, be sure not to provide boundaries that cause an invalid date selection (i.e. the 30th of every month does not include February).

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 **"Start Exchange Data Schedule" on page 16** and **"Stop Exchange Data Schedule" on page 17** 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**. See **"External Connection Establishment Function" on page 30** for more information.

Required Values

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

Up Timeout

Description

Specifies the number of seconds that the e*Way waits between calls to the **External Connection Verification function** to verify that the connection is still up. See **"External Connection Verification Function" on page 30** for more information.

Required Values

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

Resend Timeout

Description

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

Required Values

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

Zero Wait Between Successful Exchanges

Description

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

Required Values

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

See **"Exchange Data with External Function" on page 29** for more information.

3.1.3 Monk Configuration

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

Conceptually, an e*Way is divided into two halves. One half of the e*Way (shown on the left in Figure 2 below) handles communication with the external system; the other half manages the Collaborations that process data and subscribe or publish to other e*Gate components.

Figure 2   e*Way Internal Architecture
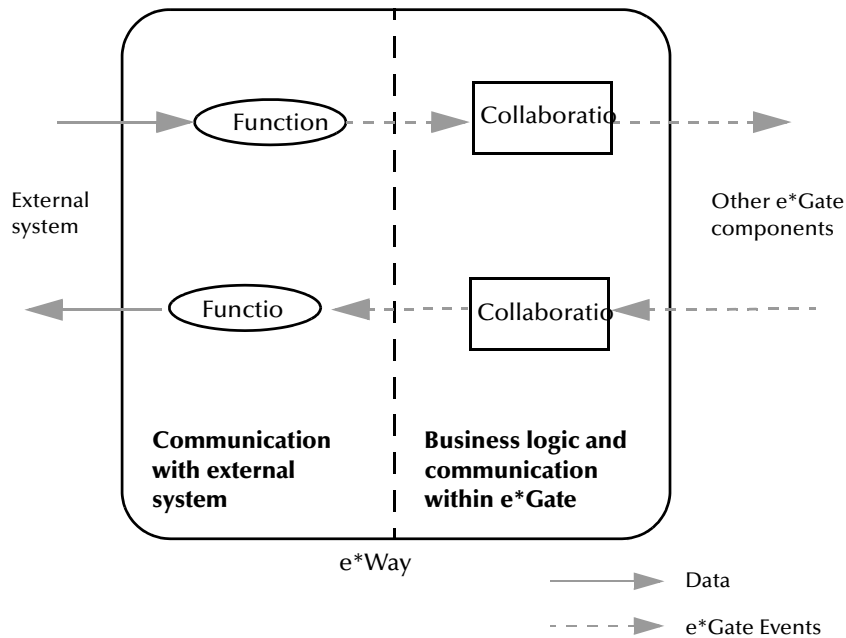
The "communications half" of the e*Way uses Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events" and send those Events to Collaborations, and manage the connection between the e*Way and the external system. The **Monk Configuration** options discussed in this section control the Monk environment and define the Monk functions used to perform these basic e*Way operations. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as **write**, **notepad**, or UNIX **vi**).

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

Operational Details

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

Type of Operation	Name
Initialization	**Startup Function** on page 28 (also see **Monk Environment Initialization File** on page 27)

Type of Operation	Name
Connection	**External Connection Establishment Function** on page 30 **External Connection Verification Function** on page 30 **External Connection Shutdown Function** on page 31
Schedule-driven data exchange	**Exchange Data with External Function** on page 29 **Positive Acknowledgment Function** on page 31 **Negative Acknowledgment Function** on page 32
Shutdown	**Shutdown Command Notification Function** on page 33
Event-driven data exchange	**Process Outgoing Event Function** on page 28

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

Initialization Functions

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

Figure 3   Initialization Functions

```
          ╭─────────────────╮
          │   Start e*Way   │
          ╰─────────────────╯
                   │
                   ▼
          ┌─────────────────┐
          │      Load       │
          │ "Monk Initialization" │
          │      file       │
          └─────────────────┘
                   │
                   ▼
          ┌─────────────────┐
          │ Execute any Monk function │
          │ having the same name as │
          │ the initialization file │
          └─────────────────┘
                   │
                   ▼
          ┌─────────────────┐
          │ Load "Startup" file │
          └─────────────────┘
                   │
                   ▼
          ┌─────────────────┐
          │ Execute any Monk function │
          │ having the same name as │
          │ the startup file │
          └─────────────────┘
```
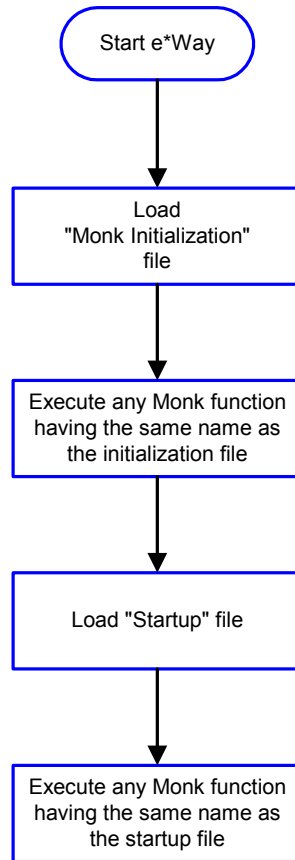
Connection Functions

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

Figure 4   Connection establishment and verification functions
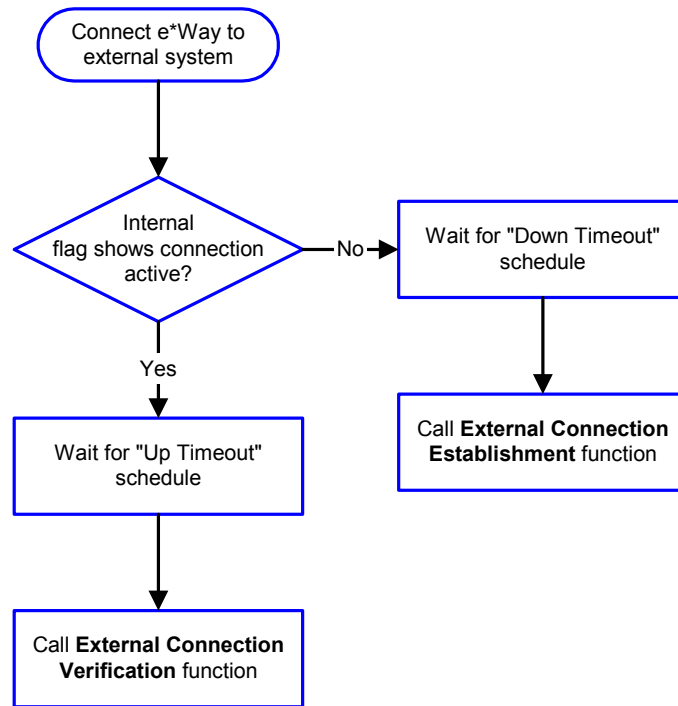
Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 6 on page 24** *and* **Figure 8 on page 26** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 101 and* **send-external-down** *on page 102 for more information.*

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

Figure 5   Connection shutdown function
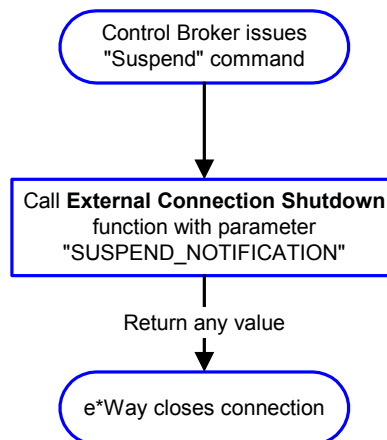
Schedule-driven Data Exchange Functions

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

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

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

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

Figure 6 Schedule-driven data exchange functions
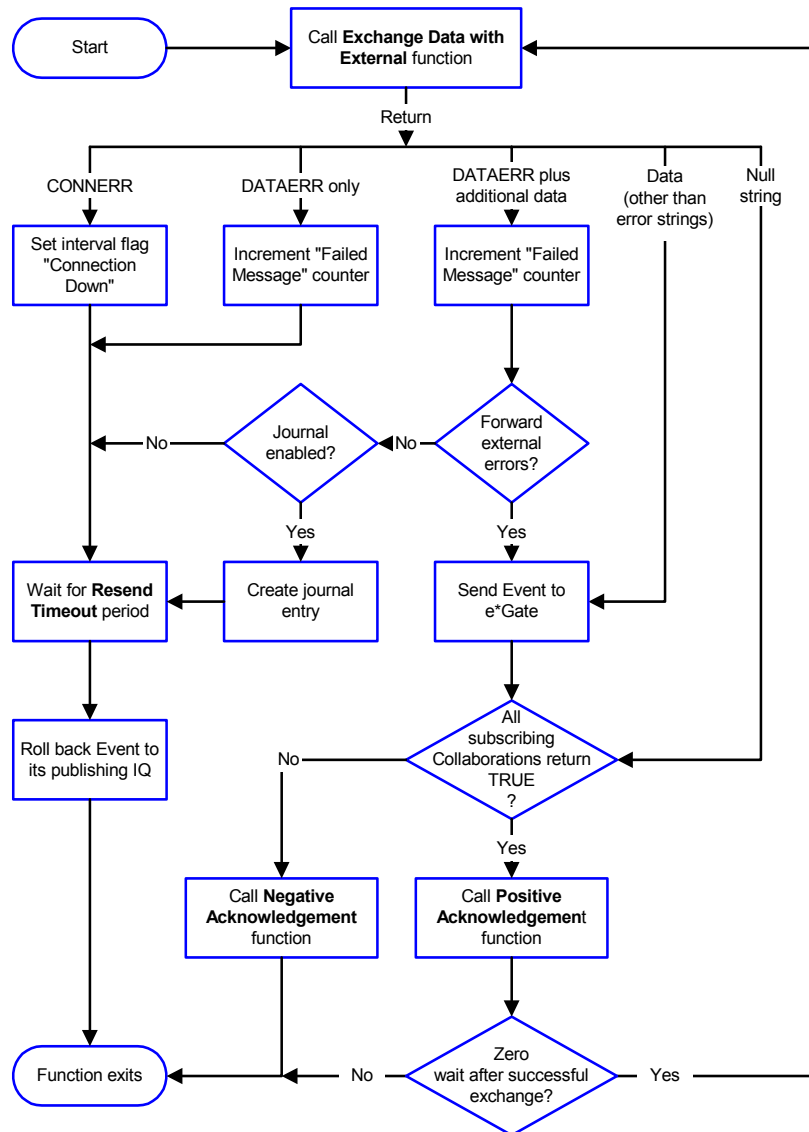
Shutdown Functions

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

Figure 7   Shutdown functions
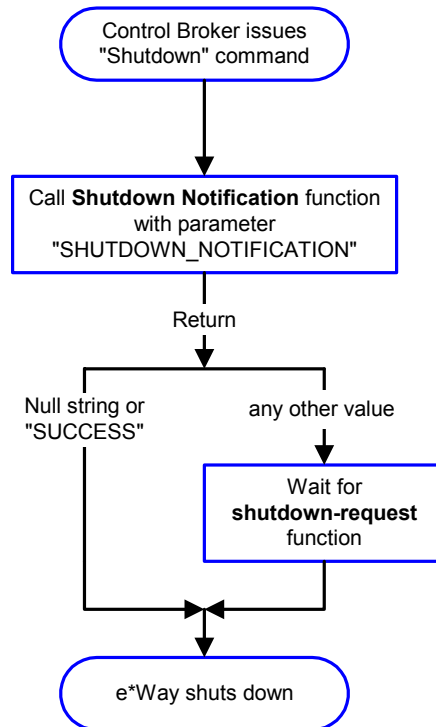
Event-driven Data Exchange Functions

Figure 8 on the next page illustrates event-driven data-exchange using the **Process Outgoing Message Function**.

Every two minutes, the e*Way checks the "Failed Message" counter against the value specified by the **Max Failed Messages** parameter. When the "Failed Message" counter exceeds the specified maximum value, the e*Way logs an error and shuts down.

Figure 8   Event-driven data-exchange functions
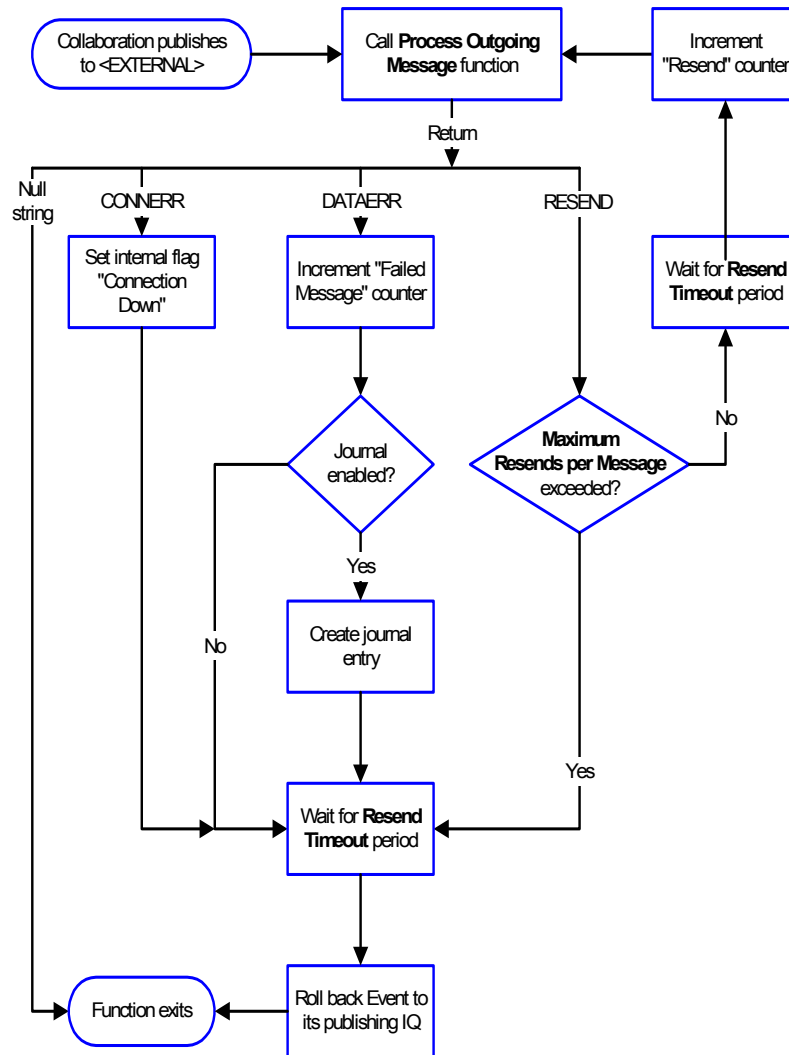
How to Specify Function Names or File Names

Parameters that require the name of a Monk function accepts 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 added 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 before the default load path.

Required Values

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

Additional information

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

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

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

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

Auxiliary Library Directories

Description

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

Required Values

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

Additional information

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

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

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

This parameter is optional and may be left blank.

Monk Environment Initialization File

Specifies a file that contains environment initialization functions, which are loaded after the auxiliary library directories are loaded. Use this feature to initialize any global Monk variables that are used by the Monk Extension 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 27** for more information about the "load path."(The default is **db-stdver-init** on page 83.)

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 try to invoke a function of the same base name as the file name (for example, for a file named **my-init.monk**, the e*Way 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 3 on page 21**).

Startup Function

Description

Specifies a Monk function that the e*Way loads and invokes upon startup, or whenever the e*Way's configuration changes before it enters into its initial communication state. This function is used so that the external system can be initialized before message exchange starts.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. (The default is **db-stdver-startup** on page 84.)

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 tries to invoke a function of the same base name as the file name (see **Figure 3 on page 21**). For example, for a file named **my-startup.monk**, the e*Way attempts to execute the function **my-startup**.

Process Outgoing Event Function

Description

Specifies the Monk function responsible for processing outgoing Event information from the e*Way to the external system when the e*Way is configured as outbound. This function is used as an event driven function.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. *You may not leave this field blank.* (The default is **db-stdver-proc-outgoing** on page 92 or **db-stdver-proc-outgoing-stub** on page 94**.**)

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 Enterprise Manager). The function returns one of the following (see **Figure 8 on page 26** for more details):

- Null string: Indicates that the Event was published successfully to the external system.

- "RESEND": Indicates that the Event should be resent.

- "CONNERR": Indicates that there is a problem communicating with the external system.

- "DATAERR": Indicates that there is a problem with the message (Event) data itself.

- If a string other than the following is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function.

Note: *If you want to use **event-send-to-egate** to enqueue failed Events in a separate IQ, the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See the Monk Developer's Reference for more information.*

Exchange Data with External Function

Description

Specifies a Monk function that initiates an exchange of data with an external system that can be either inbound or outbound. This function is called according to a schedule (unlike the **Process Outgoing Message Function**, which is event-driven), predominantly inbound.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. (The defaults are **db-stdver-data-exchg** on page 96 or **db-stdver-data-exchg-stub** on page 97.)

Additional Information

The function accepts no input and must return a string (see **Figure 6 on page 24** for more details):

- Null string: Indicates that the data exchange was completed successfully. No information is sent into the e*Gate system.

- "CONNERR": Indicates that a problem with the connection to the external system has occurred.

- "DATAERR": Indicates that a problem with the data itself has occurred. The e*Way handles the string "DATAERR" and "DATAERR" plus additional data differently; see **Figure 6 on page 24** for more details.

- Any other string: The contents of the string are packaged as an inbound Event. The e*Way must have at least one Collaboration configured suitably to process the inbound Event, as well as any required IQs.

This function is initially triggered by the **Start Data Exchange** schedule or manually by the Monk function **start-schedule**. After the function has returned true and the data received by this function has been ACKed or NAKed (by the **Positive Acknowledgment Function** or **Negative Acknowledgment Function**, respectively), the e*Way checks the **Zero Wait Between Successful Exchanges** parameter. If this parameter is set to **Yes**, the e*Way immediately calls the **Exchange Data with External** function again; otherwise, the e*Way does not call the function until the next scheduled "start exchange" time or the schedule is manually invoked using the Monk function **start-schedule** (see **start-schedule** on page 99 for more information).

External Connection Establishment Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***This field cannot be left blank***.(The default is **db-stdver-conn-estab** on page 85.)

Additional Information

The function accepts no input and must return a string:

- "SUCCESS" or "UP": Indicates that the connection was established successfully.

- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

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

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

External Connection Verification Function

Description

Specifies a Monk function that the e*Way calls to confirm that the external system is operating and available.

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. (The default is **db-stdver-conn-ver** on page 87.)

Additional Information

The function accepts no input and must return a string:

- "SUCCESS" or "UP": Indicates that the connection was established successfully.

- Any other string (including the null string): Indicates that the attempt to establish the connection failed.

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

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

External Connection Shutdown Function

Description

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

Required Values

The name of a Monk function. (The default is **db-stdver-conn-shutdown** on page 88.)

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, and passes 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.

Note: *Include in this function any required "clean up" that must be performed as part of the shutdown procedure, but before the e*Way exits.*

Positive Acknowledgment Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined. (The default is **db-stdver-pos-ack** on page 89.)

Additional Information

The function requires a non-null string as input, and returns a string.

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

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the Positive Acknowledgment function is called again, with the same input data.

- Null string: The function completed execution successfully.

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

Note: *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

Negative Acknowledgment Function

Description

Specifies a Monk function 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. (The is default is **db-stdver-neg-ack** on page 90.)

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.

- Null string: The function completed execution successfully.

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

Note: *If you configure the acknowledgment function to return a non-null string, you must configure a Collaboration (with appropriate IQs) to process the returned Event.*

The e*Way exits if it fails its attempt to invoke this function, or this function returns a **FAILURE** string.

Shutdown Command Notification Function

Description

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

Required Values

The name of a Monk function. (The default is **db-stdver-shutdown** on page 91.)

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:

- A null string or "SUCCESS": Indicates that the shutdown can occur immediately.

- Any other string: Indicates that shutdown must be postponed. Once postponed, shutdown does not proceed until the Monk function **shutdown-request** is executed (see **shutdown-request** on page 105).

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

3.1.4 Database Setup

Database Type

Description

Specifies the type of database.

Required Values

DB2, **ODBC**, **ORACLE8**, **ORACLE8i**, **SYBASE11**, or **SYBASE12**

Note: *Any other value is effectively equal to ODBC.*

Database Name

Description

This is the name of the Server entry name from the Sybase interfaces file. To access a particular Sybase database, either make that database the default for the user on the database server, or issue a "use database" from the Monk scripts.

Required Values

Any valid string.

User Name

Description

The name used to access the database.

Required Values

Any valid string.

Encrypted Password

Description

The password that provides access to the database.

Required Values

Any valid string.

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

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

Implementation

This chapter contains information explaining the use of the ETD Editor's Build Tool as well as two sample Sybase e*Way scenarios.

This Chapter Includes:

- **"Using the ETD Editor's Build Tool" on page 35**
- **"Sample One – Event Driven" on page 45**
- **"Sample Two – Schedule Driven Database Access" on page 61**
- **"Sample Monk Scripts" on page 65**

4.1 Using the ETD Editor's Build Tool

The Event Type Definition Editor's Build tool automatically creates an Event Type Definition file based on the tables in an existing database. The Event Type Definition (ETD) can be created based on one of (or a combination of) the following criteria:

- **Table or View** – Displays all of the columns in the specified table or view.
- **Dynamic SQL Statement** – Displays the format of the results of an SQL statement. This can be used to return only a few of the columns in a table.
- **Stored Procedure** – Displays the format of the results of an SQL Stored Procedure. This option is only available for *Delimited* messages. Please note that the Sybase e*Way does not support user defined data types.

The results of these three types of message criteria are explained in **"The Event Type Definition Files" on page 38**.

To create an Event Type Definition using the Build Tool:

1 Launch the ETD (Event Type Definition) Editor.

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

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

3 In the File name box, type the name of the ETD file you wish to build. *Do not specify any file extension*—the Editor will supply an "ssc" extension for you.

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

5   Under **Select a Library Converter**, select DART Converter.

6   Click **OK**.

7   The Converter Wizard launches.

Figure 9   Converter Wizard Subordinate Dialog Box
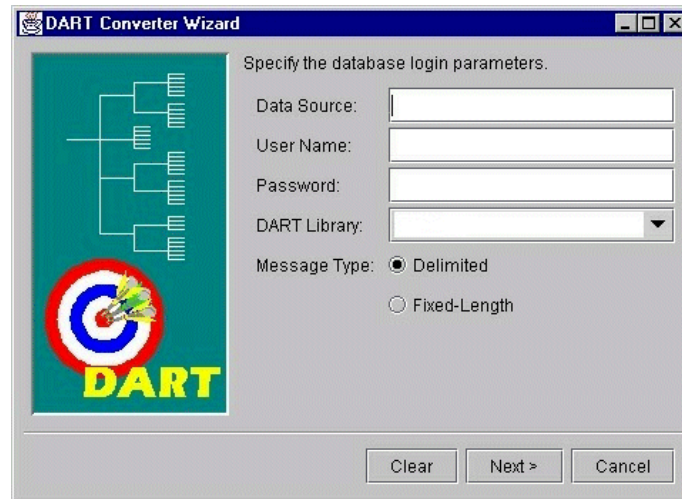
8   Enter the Data Source.

9   Enter the User Name.

10   Enter the Password.

11   Select the DART Library. Prior to selecting the appropriate library, you will need to install the corresponding e*Way.

12   Select the correct Message Type.

If you select the Delimited Message Type, the following dialog box will appear.

Figure 10   Converter Wizard Delimited Message Type Dialog Box
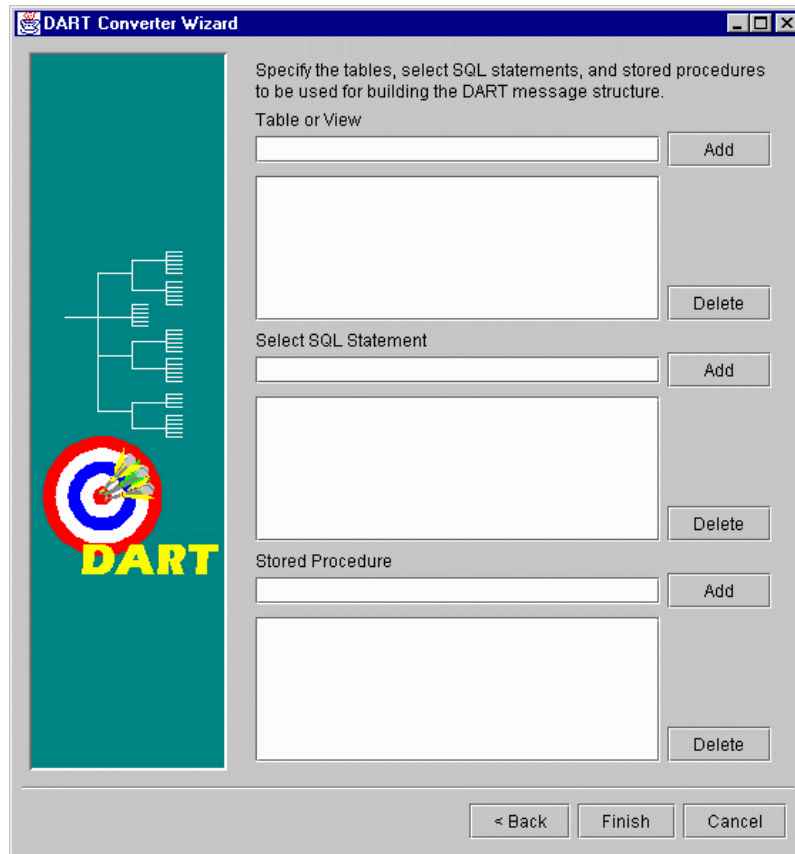
13 Select or Add the correct Table or View.

14 Select or Add the correct SQL Statement.

15 Edit or Finish your selections.

If you select the Fixed-Length Message Type, the following dialog box will appear.

Figure 11 Converter Wizard Fixed-Length Message Type Dialog Box
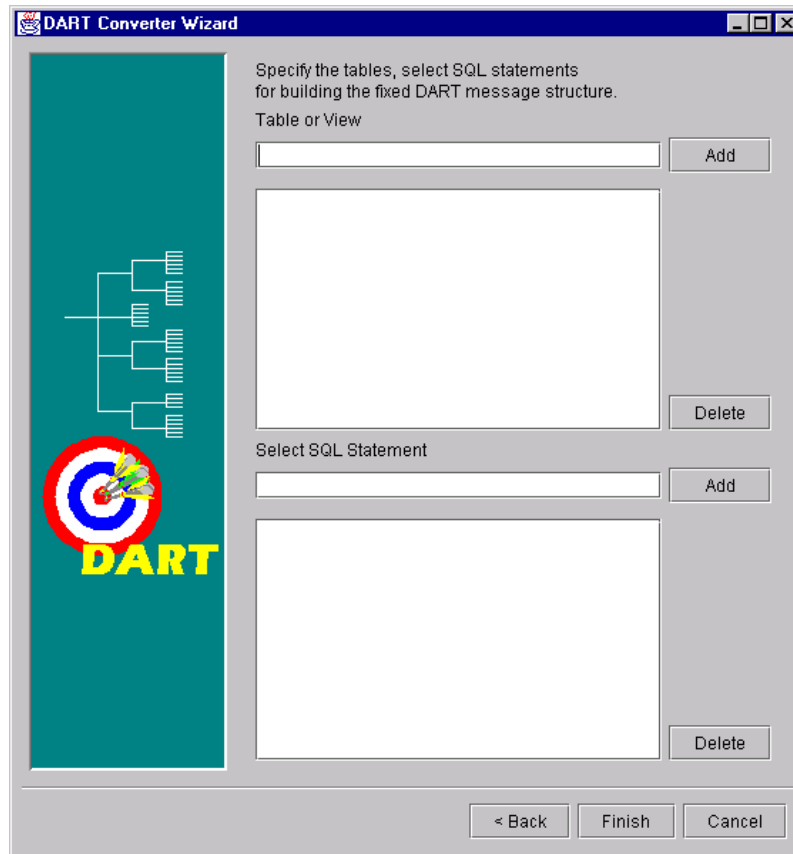
16 Select or Add the correct Table or View.

17 Select or Add the correct SQL Statement

18 Edit or Finish your selections.

Note: *The (#) character cannot be used in the node name of the .ssc file. The Sybase e*Way will be unable to generate the correct node name for the column name of a table that contains the (#) character, as Monk will filter out the character.*

4.1.1 The Event Type Definition Files

The DART Converter Build Tool will create a different ETD based on the criteria that was specified in the Build Tool Wizard (see **Figure 9 on page 36** and **Figure 10 on page 37**).

Table or View

Entering a table or view name as a selection criteria will display all of the columns in that table or view. This is useful when you want to access an entire record from the table as an e*Gate Event. The criteria shown in Figure 12 generates the ETD shown in Figure 13.

Figure 12 Table or View Selection
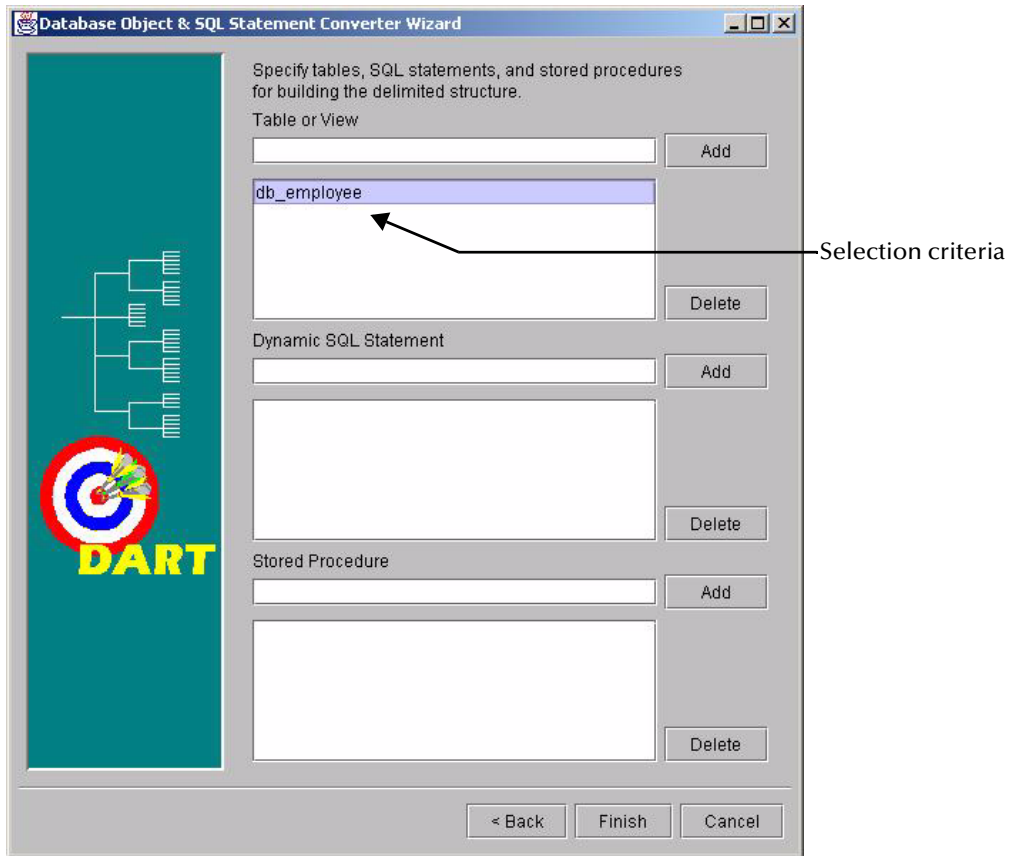
Figure 13 Table or View ETD
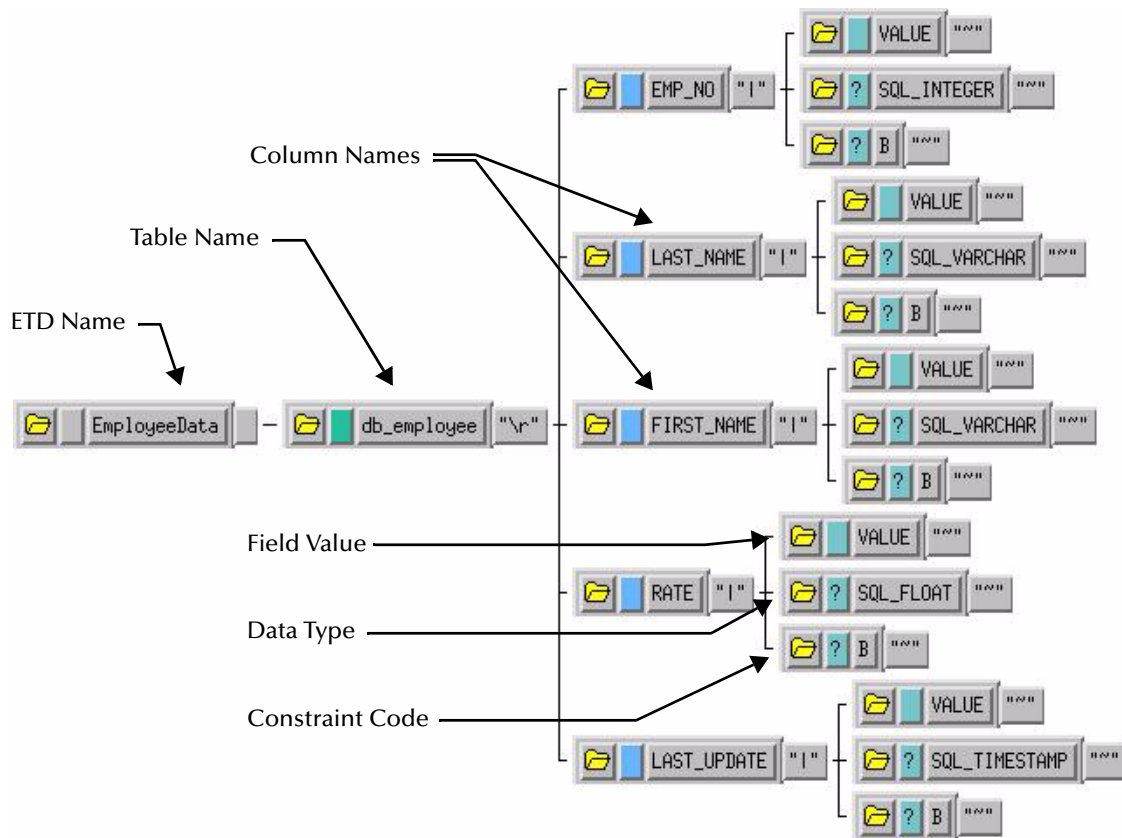
The ETD that is generated by the DART Converter Build Tool using the Table or View criteria contains the elements shown in the table below.

Table 2 Elements of the Table or View ETD

Element	Description
ETD Name	This is the root node of the Event Type Definition.
Table Name	This node displays the name of the table or view.
Column Name	This is the name of the column(s) in the selected table or view.
Field Value	This is the value of the data in the column. This can be thought of as the *payload data* for this column.
Data Type	This node designates the type of data contained in the value field.
Constraint Code	The constraint codes are based on the column constraints in the table. The possible codes are: ▪ **I** – *Insert* operations are allowed in this column. ▪ **U** – *Update* operations are allowed in this column. ▪ **N** – *Neither* insert nor update operations are allowed in this column. ▪ **B** – *Both* insert and update operations are allowed in this column.

Dynamic SQL Statement

Entering an SQL statement as a selection criteria will display the format of the results of that SQL statement. This is useful when you only want to access certain columns from the table for a particular e*Gate Event.

To use this type of ETD, you should use the **db-stmt-bind** function to bind the dynamic statement and **db-struct-execute** function to execute the SQL statement. For more information, see **db-stmt-bind** on page 138 and **db-struct-execute** on page 182.

The SQL statement shown in Figure 14 generates an ETD that returns specific records from the table based on the selection criteria (which is represented by a question mark "?"). The resulting ETD is shown in Figure 15.

Note: *It is not necessary to include the terminating semi-colon as part of the SQL statement.*

Figure 14 Dynamic SQL Statement Selection
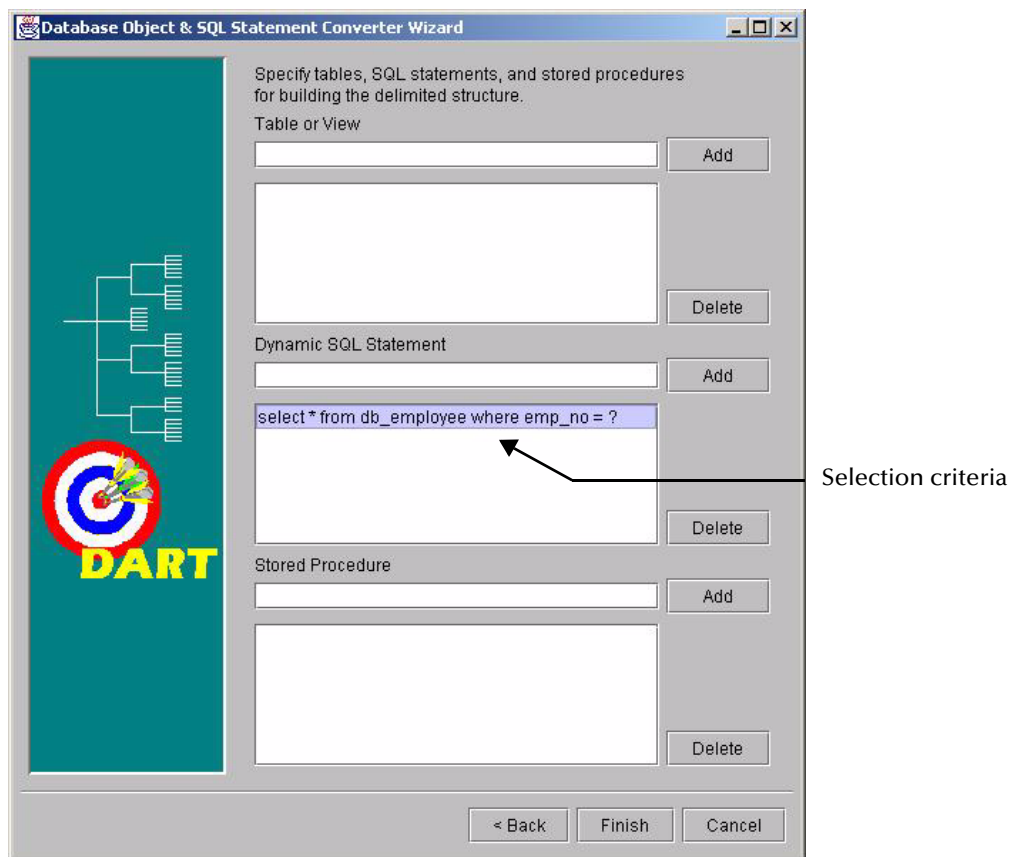
Figure 15 Dynamic SQL Statement ETD
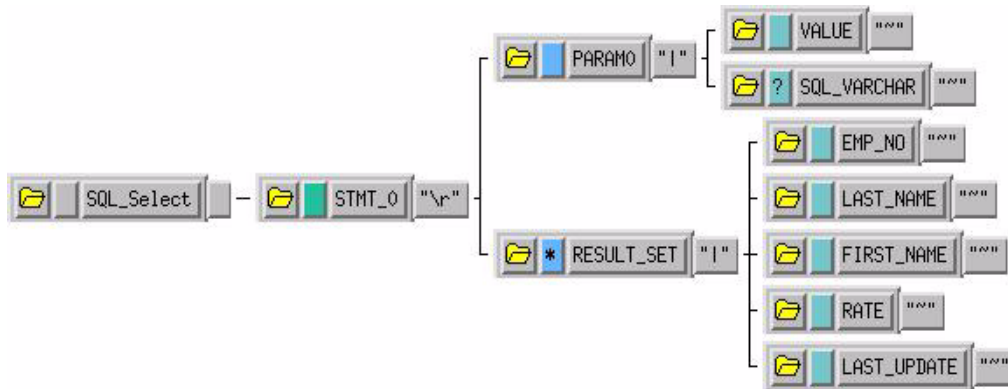
The **PARAM0** node in the ETD shown in Figure 15 represents the criteria specified in the SQL statement. Additional criteria would be represented in additional nodes (**PARAM1**, **PARAM2**, and so forth). For example, using the following SQL statement:

```
SELECT * FROM db_employee WHERE last_name = ? AND first_name = ?
```

the Build Tool would generate an ETD with two input parameter nodes (**PARAM0** and **PARAM1**)—one for each of the criteria (?). The **VALUE** nodes of these input parameter nodes are used to carry the payload of the selection statement.

Stored Procedure

Entering a stored procedure name as a selection criteria will generate an ETD that will access a stored procedure in the external database. This is useful when you want to access the results of a stored procedure.

The stored procedure specified in Figure 16 generates an the ETD shown in Figure 17. Below is the contents of the sample stored procedure:

```
procedure GET_EMPLOYEES
(
    dept_number in      integer,
    batch_size  in      integer,
    found       in out  integer,
    done_fetch  out     integer,
    emp_name    out     charArrayTyp,
    job_title   out     charArrayTyp,
    salary      out     numArrayTyp
) is
begin
    if not get_emp%isopen then
        open get_emp(dept_number);
    end if;
    done_fetch := 0;
    found := 0;
    for i in 1..batch_size loop
        fetch get_emp into emp_name(i),
            job_title(i), salary(i);
        if get_emp%notfound then
            close get_emp;
            done_fetch := 1;
            exit;
        else
            found := found + 1;
        end if;
    end loop;
end get_employees;
```

Figure 16 Stored Procedure Selection
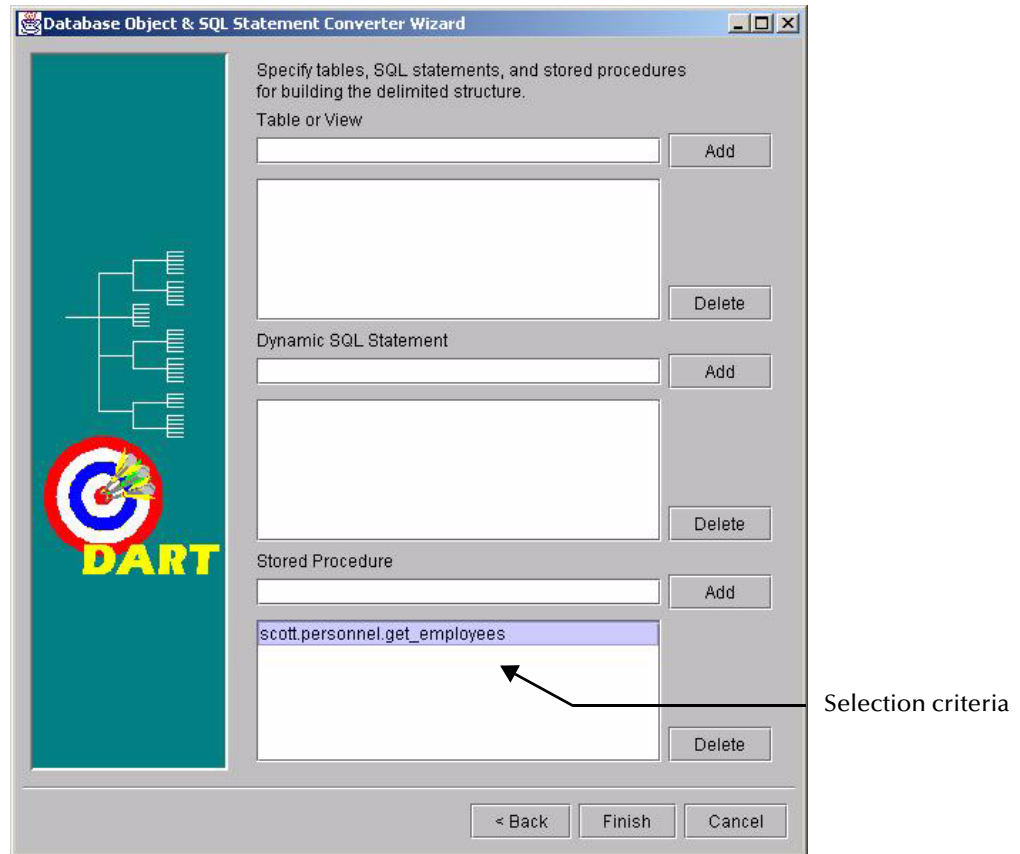
Note: *Although periods can be entered in the selection criteria in the Build Tool, they are not permitted in the node names of the ETD. Any periods in the selection criteria will be converted to asterisks in the generated ETD. See Figure 17.*

Figure 17   Stored Procedure ETD
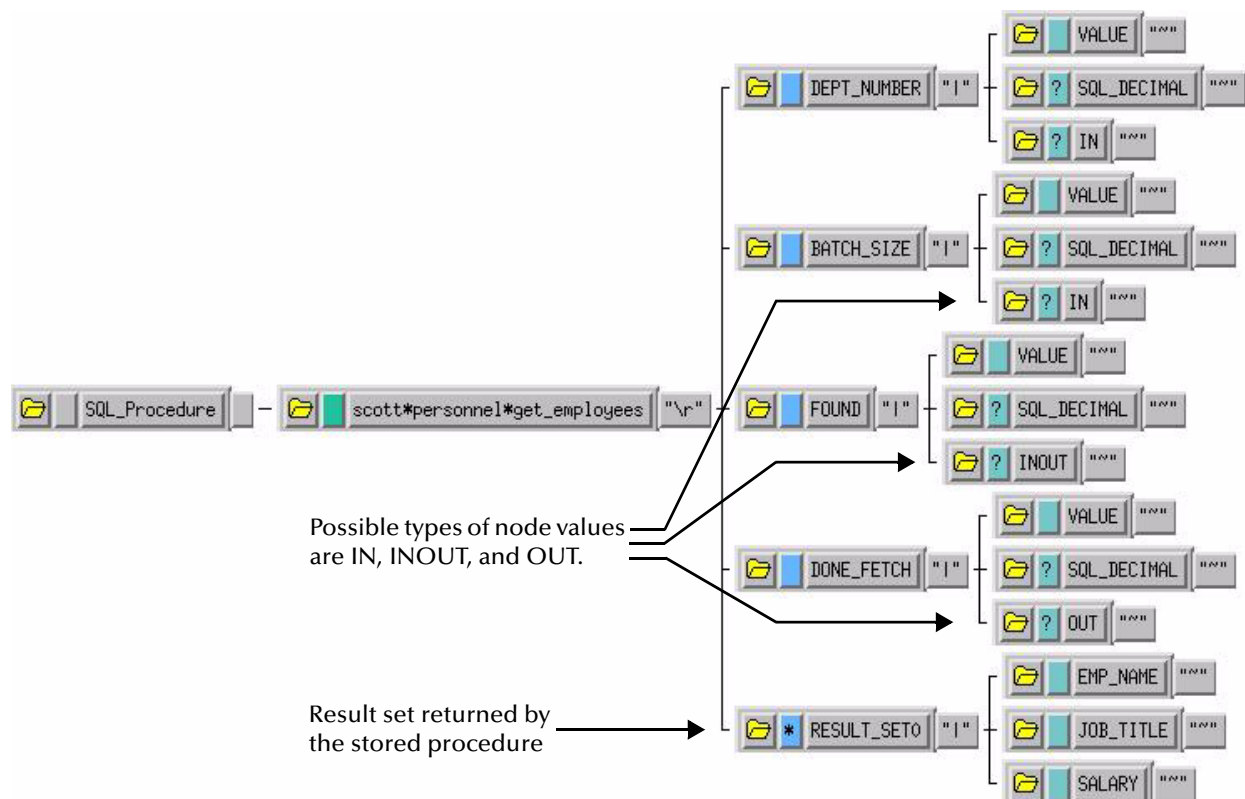
This Event Type Definition is used to pass certain input to the stored procedure. The nodes with types of **IN** or **INOUT** are used as input. The nodes with types of **OUT** or **INOUT** can be used for output. The results of the stored procedure are returned to the **RESULT_SET0** node. The Build Tool will create additional result set nodes (**RESULT_SET1**, **RESULT_SET2**, and so forth) for stored procedures returning multiple results.

4.2   Sample One – Event Driven

The previous sections provided the basics for implementing the Sybase e*Way. This section describes how to use the Sybase e*Way within a sample schema. This sample will read an input file, apply associated Collaborations and Rules, and then output the results in a specified location. It is assumed that the Database is installed and configured properly.

This implementation will consist of two e*Ways, three Event Types, two Collaboration Rules, an Intelligent Queue (IQ), and two Collaborations, as follows:

- **FileIn** - This e*Way will read sample inbound data and copy it to the IQ.

- **dart_rcv** - This e*Way applies a Collaboration Rule to data, and transforms the data.

- **db_rcv_in** - This Event Type processes inbound data copied to an IQ.

- **db_rcv_struct** - This Event Type defines the format of data written out from a Collaboration.

- **FileInEvent** - This Event Type describes data that is input from an external source to a Collaboration.

- **db_rcv** - This Collaboration Rule is associated with *db_rcv_struct* for output Event Types, and *db_rcv_in* for input Event Types.

- **no_xlate** - The Collaboration Rule is associated with *FileInEvent* Event Type for input, and the *db_rcv_in* Event Type for output.

- **IQ1** - This Intelligent Queue is a Standard STC IQ.

- **Pub** - This Collaboration will be a member of the *FileIn* e*Way, applying the *no_xlate* Collaboration Rule, and will contain the *FileInEvent* and *db_rcv_in* Event Types for input and output, respectively.

- **Sub** - This Collaboration will be a member of the *dart_rcv* e*Way, applying the *db_rcv Collaboration* Rule, and will contain the *db_rcv_in* and *db_rcv_struct* Event Types for input and output, respectively.

Figure 18   Sample Schema Data Flow
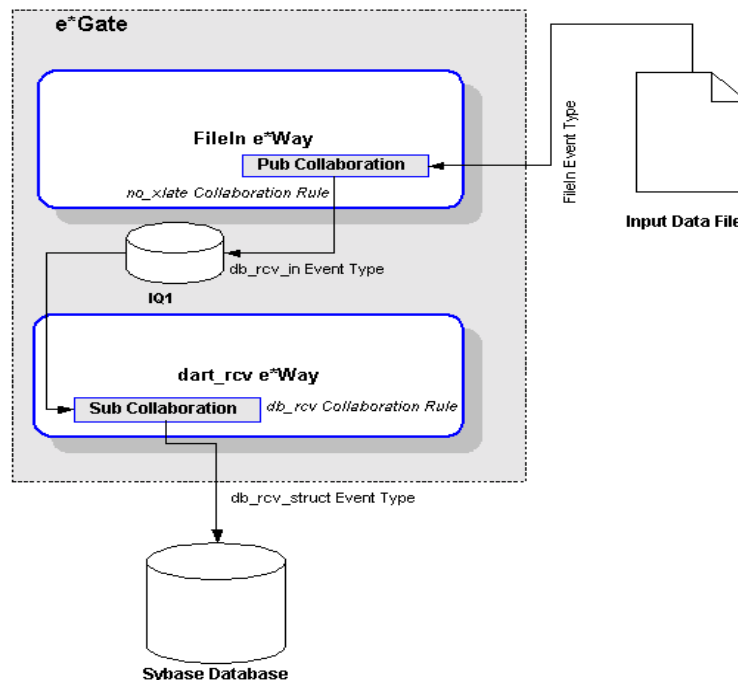
This sample schema will process data from a file, apply Collaboration Rules, and send the data to the database. The sample will also serve to verify that the Sybase Intelligent Adapter has been properly installed and configured. The following sections provide the specifics for implementing the sample.

4.2.1 **Creating the New Schema**

The first task in deploying the sample implementation is to create a new Schema name. While it is possible to use the default schema for the sample implementation, it is recommended that you create a separate schema for testing purposes. After you install the Sybase e*Way Intelligent Adapter, do the following:

1 Start the e*Gate Enterprise Manager GUI.

2 When the Enterprise Manager prompts you to login, select the host that you specified during installation, and enter your password.

3 You will then be prompted to select a schema. Click on **New**.

4 Enter a name for the new Schema; In this case, enter *Sybase_Test*, or similar name as desired.

The e*Gate Enterprise Manager opens under your new schema. You are now ready to begin creating the necessary components for this sample schema.

4.2.2 **Creating the Event Types**

The next step is creating the three event types mentioned previously. To create an Event Type, you may use the ETD Editor build tool (See the section **"Using the ETD Editor's Build Tool" on page 35**), or you can open the ETD Editor and add the root node and subnodes as needed.

The three Event Types in this sample are:

- FileInEvent

- db_rcv_in

- db_rcv_struct

Before creating the Collaboration Rules, assure your default Collaboration Editor is set to Java. To do this do the following:

1 From the e*Gate Enterprise Manager toolbar, click **Options**.

2 Click **Default Editor...**

3 Select **Monk**.

4 Click **OK**.

FileInEvent

This Event Type will process inbound data from an external file. *FileInEvent* will use the *EventMsg.ssc* file that is included with the Sybase Intelligent Adapter e*Way installation and located at the default e*Gate */monk_scripts/common* directory. To create *FileInEvent*, do the following:

1 Highlight the Event Types folder on the Components tab of the e*Gate Navigator.

2 On the Palette, click .

3 Enter *FileInEvent* as the name, then click **OK.**

4 Select the *FileInEvent*, then click [icon] to edit its properties.

5 When the **Properties** window opens, click on the **Find** button, and select *EventMsg.ssc*.

6 Click **OK**.

db_rcv_in

This Event Type represents the data transported by the *FileIn* e*Way and processed by the *Pub* Collaboration. To create this Event Type do the following:

1 Highlight the Event Types folder on the Components tab of the e*Gate Navigator.

2 On the Palette, click [icon].

3 Enter *db_rcv_in* as the name, then click **OK.**

4 Select *db_rcv_in*, then click [icon] to edit its properties.

5 When the Properties window opens, click on the **New** button.

When the ETD Editor opens, add the root node and the subnodes. When you are finished, the file should be similar to the following:

Figure 19 Sample ETD for db_rcv_in
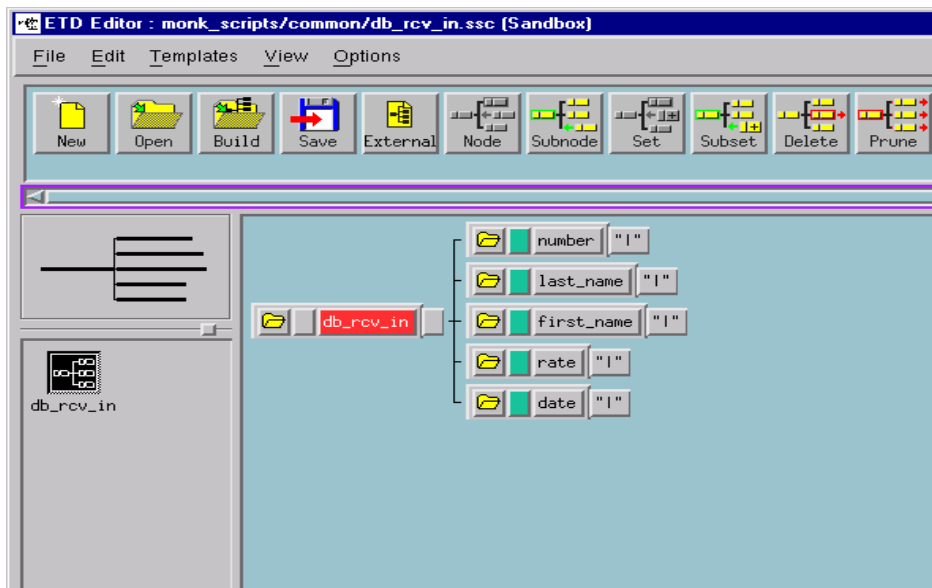
6 Save this file as *db_rcv_in.ssc*, and promote it to runtime, and exit from the ETD Editor.

7 Click OK in the Event Type Properties window to return to the e*Gate Enterprise Manager GUI.

db_rcv_struct

This Event Type represents the transformed data from the *db_rcv_in* Event Type, and transported through the *dart_rcv* e*Way to the external Sybase database.

1 Highlight the Event Types folder on the Components tab of the e*Gate Navigator.

2 On the Palette, click ▦.

3 Enter *db_rcv_struct* as the name, then click **OK.**

4 Select *db_rcv_struct*, then click ▦ to edit its properties.

5 When the Properties window opens, click on the **New** button.

   When the ETD Editor opens, add the root node and the subnodes. When you are finished, the file should be similar to the following:

Figure 20   Sample EDT for db_rcv_struct
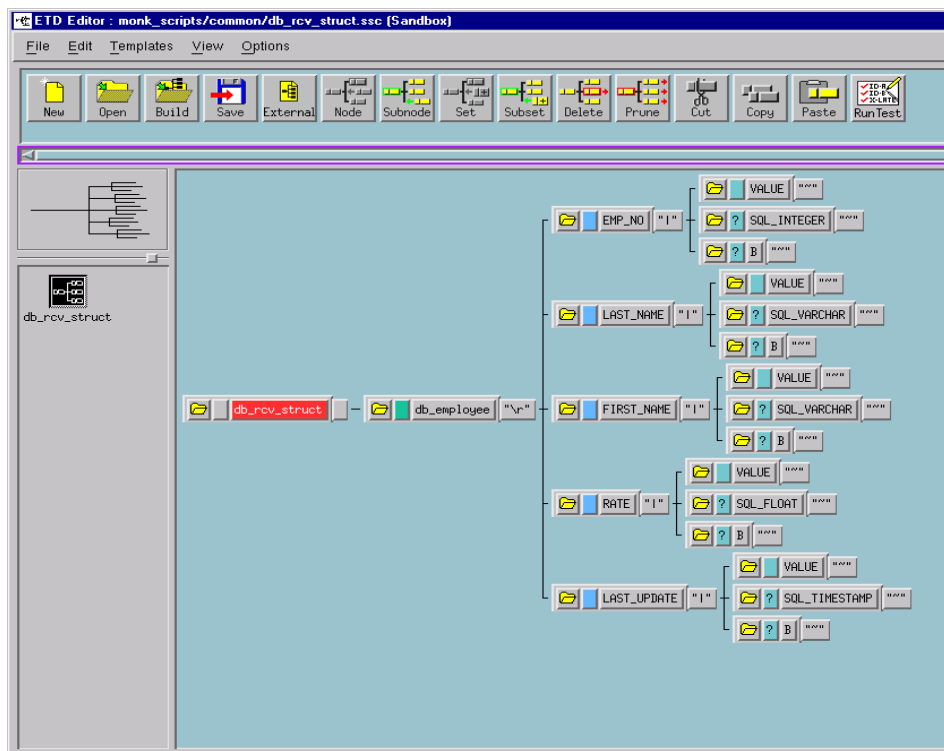
4.2.3 Creating and Configuring the e*Ways

The next step in implementing this sample schema is to create two e*Ways:

- *FileIn*

- *dart_rcv*

Details for creating each are provided in the following sections.

FileIn

This e*Way will receive data from an external file, apply a Collaboration Rule to transform the data, and then publish it to the *IQ1* Intelligent Queue. To create *FileIn*, do the following:

1 Select the Navigator's Components tab.

2 Open the host on which you want to create the e*Ways.

3 Select the Control Broker that will manage the new e*Way.

4 On the Palette, click ▣.

5 Enter the name of the new e*Way, (in this case, *FileIn*), then click **OK**.

6 Select *FileIn*, then click ▣ to edit its properties.

7 When the e*Way Properties window opens, click on the **Find** button beneath the *Executable File* field, and select *stcewfile.exe* for the executable file.

8 Under the Configuration File field, click on the **New** button. When the Settings page opens, set the following for this configuration file:

Table 3 Configuration Parameters for FileIn e*Way

Parameter	Value
General Settings	
AllowIncoming	Yes
AllowOutgoing	No
Outbound Settings	Default
Poller Inbound Settings	
PollDirectory	/egate/data (input file folder)
InputFileExtension	*.demodat (input file extension)
PollMilliseconds	Default
Remove EOL	Default
MultipleRecordsPerFile	Default
MaxBytesPerLine	Default
BytesPerLineIsFixed	Default

9 Exit from **Settings**, then save and promote the file as *dart_inbound.cfg*.

dart_rcv

This e*Way will receive the transformed data from *IQ1*, then forward it to the Sybase database. Before you can use the e*Gate Enterprise Manager GUI to create *dart_rcv*, it is required that you first compose a DART (*.dsc) script file. You may use any ASCII text editor, or you may use the E*Gate Enterprise Manager's Collaboration Rules Editor.

Using an ASCII Text Editor

If you use an ASCII text editor, the file you create should be similar to the sample shown as follows:

```
;;
DART-mode: RECEIVE
;; source-event-path: monk_scripts/common/db_rcv_in.ssc
;; destination-event-path: monk_scripts/common/db_rcv_struct.ssc
(define usercomment "")
(define version "3.1")
(define xlate-name "db_rcv")
(define input-message-format-file-name "db_rcv_in.ssc")
(define output-message-format-file-name "db_rcv_struct.ssc")
(load "db_rcv_in.ssc")
(load "db_rcv_struct.ssc")
(define src-collapsed-nodes '(
))
(define dest-collapsed-nodes '(
))
(define collapsed-rules '(
))
(define db_rcv
  (let ((input ($make-event-map db_rcv_in-delm db_rcv_in-struct))
        (output ($make-event-map db_rcv_struct-delm db_rcv_struct-
struct))
       )
    (lambda (message-string)
      ($event-parse input message-string)
      ($event-clear output)
      (begin
       (display "LOAD PATH: ")
       (display load-path) (newline)
       (newline)
       (copy-strip ~input%db_rcv_in.number
~output%db_rcv_struct.db_employee.EMP_NO.VALUE "")
       (copy-strip ~input%db_rcv_in.last_name
~output%db_rcv_struct.db_employee.LAST_NAME.VALUE "")
       (copy-strip ~input%db_rcv_in.first_name
~output%db_rcv_struct.db_employee.FIRST_NAME.VALUE "")
       (copy-strip ~input%db_rcv_in.rate
~output%db_rcv_struct.db_employee.RATE.VALUE "")
       (copy-strip ~input%db_rcv_in.date
~output%db_rcv_struct.db_employee.LAST_UPDATE.VALUE "")
       (display ~output%db_rcv_struct)
       (display "READY to INSERT into DATABASE") (newline)
       (if (db-struct-insert connection-handle
~output%db_rcv_struct.db_employee)
          (begin
           (db-commit connection-handle)
           (display "Record Inserted") (newline)
          )
          (begin
           (display "Structure insert failed: ")
           (display (db-get-error-str connection-handle)) (newline)
           (if (db-check-connect)
            (begin
             (event-dataerr (get ~input%db_rcv_in))
            )
            (begin
             (event-connerr "")
            )
           )
          )
```

```
         )
      )
      (let ((result ""))
         ($event-clear input)
         ($event-clear output)
         result)
  )))
```

When you complete the DART Script, do the following:

- Save the file as *db_rcv.dsc* into the **egate/client/monk_scripts/common** directory.

- From the e*Gate Enterprise Manager GUI, select **File**, **Commit to Sandbox...**, and specify *db_rcv.dsc* so that it will be available for use by e*Gate.

Using the Collaboration Rules Editor

Alternatively, you can use the Collaboration Rules Editor to create the DART Script. To do so, follow these steps:

1 From the e*Gate Enterprise Manager GUI, click on [icon] to open the Collaboration Rules Editor.

2 From the Collaboration Rules Editor, click on [New icon].

3 When the *New* file dialog box opens, enter the following information in the corresponding fields as follows:

Figure 21 Collaboration Rules Editor **New** file dialog box
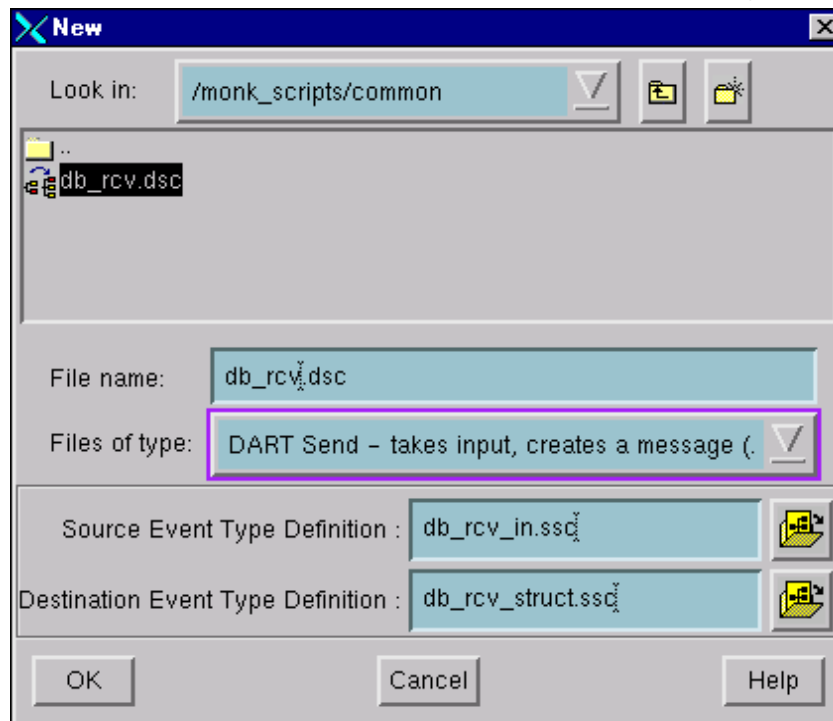
A **File name**: *db_rcv.dsc*

B **Files of type**: *DART Send - takes input, creates a message (.dsc).*

Click on [icon] to open the drop down list menu, and select *DART Send - takes input, creates a message (.dsc).*

C **Source Event Type Definition**: *db_rcv_in.ssc*

Click on [icon] to open the list of ETD files, and select *db_rcv_in.ssc.*

D **Destination Event Type Definition**: *db_rcv_struct.ssc*

Click on [icon] to open the list of ETD files, and select *db_rcv_struct.ssc.*

E Click on **OK** to return to the Collaboration Rules Editor. Your new DART script file should appear similar to the sample below.

Figure 22 Creating a DART Script with Collaboration Rules Editor
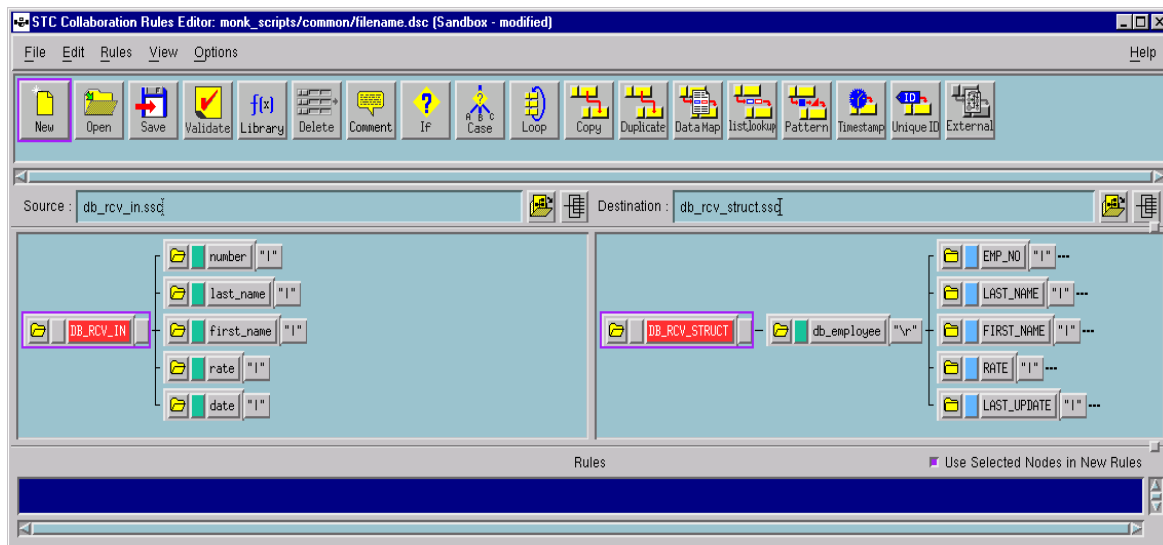
4 You need to create the rules for this file.

A Use the copy function to transform data from the source to the destination for each node.

For example, select the *number* subnode in the *Source* pane; then select the

EMP_NO.VALUE subnode, and click on [icon] .

Use the Copy function for each of the source subnodes to transform data to the destination source as follows:

Table 4   Copy Function in db_rcv.dsc

Source Subnode	Destination Subnode
number	EMP_NO.VALUE
last_name	LAST_NAME.VALUE
first_name	FIRST_NAME.VALUE
rate	RATE.VALUE
date	LAST_UPDATE.VALUE

B   You also need to add DISPLAY, FUNCTION, and IF-ELSE rules. Details about Functions to use can be found in **"Sybase e*Way Functions" on page 82**. Information about using the Collaboration Rules Editor is available in the On-line Help.

When you have finished creating the rules for this file, the Rules window in the Collaboration Rules Editor should be similar to the following example:

Figure 23   db_rcv.dsc Rules
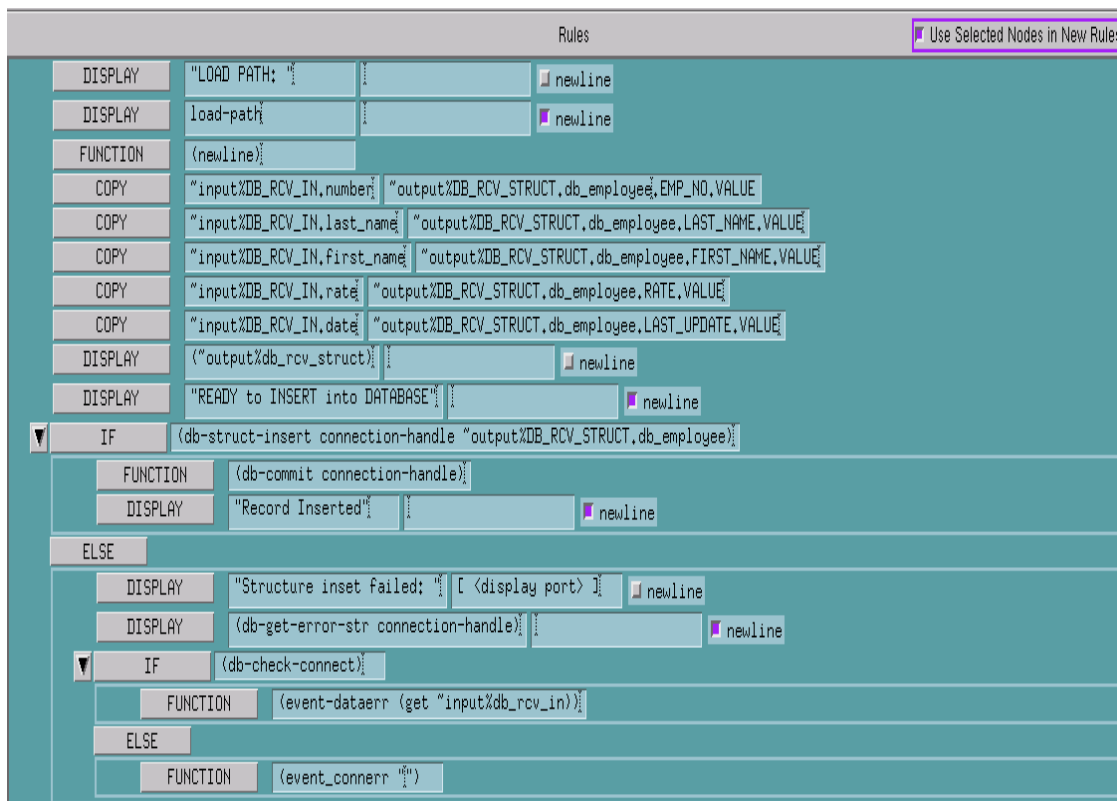
In its totality, the db_rcv.dsc file within the Collaboration Rules Editor will be similar to the following:

Figure 24 db_rcv.dsc in the Collaboration Rules Editor
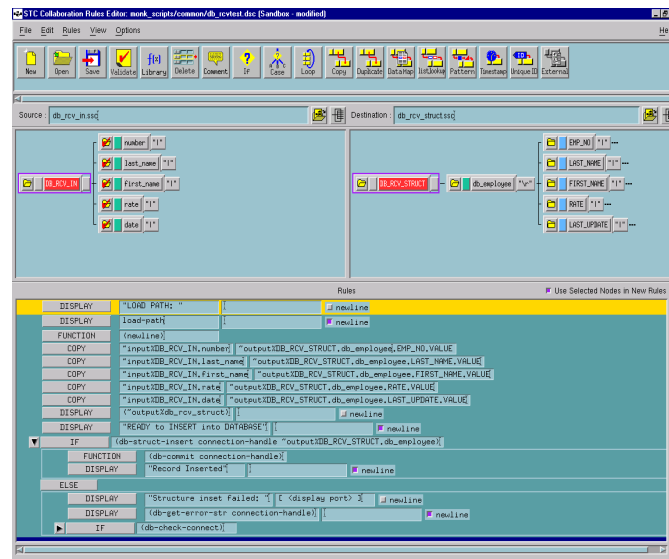
5   Select **File**, **Save**, and exit from the Collaborations Rules Editor to return to the e*Gate Enterprise Manager GUI.

Creating the dart_rcv e*Way

Next, create the *dart_rcv* e*Way as follows:

1   Select the Navigator's Components tab.

2   Open the host on which you want to create the e*Ways.

3   Select the Control Broker that will manage the new e*Way.

4   On the Palette, click [icon].

5   Enter the name of the new e*Way, (in this case, *Dart_rcv*), then click OK.

6   Select *Dart_rcv*, then click [icon] to edit its properties.

7   When the e*Way Properties window opens, click on the **Find** button beneath the *Executable File* field, and select *stcewgenericmonk.exe* for the executable file.

8   Click on **New** under the *Configuration File field*, and select *Dart* in the e*Way Template Selection window. When the Settings page opens, set the following for this configuration file:

Table 5 Parameter Settings for dart_rcv e*Way

Parameter	Value
General Settings	Default
Communication Setup	
Start Exchange Data Schedule	Repeatedly, every 1 minute
Stop Exchange Data Schedule	None

Table 5 Parameter Settings for dart_rcv e*Way

Parameter	Value
Exchange Data Interval	120
Down Timeout	Default
Up Timeout	Default
Resend timeout	Default
Zero Wait between successful Exchanges	Default
Monk Configuration	
Additional Path	Blank
Auxiliary Library Directories	monk_library/dart
Monk Environment Initialization File	db-stdver-init
Startup Function	db- stdver-startup
Process Outgoing Message Function	monk_scripts/common/db_rcv.dsc
Exchange Data With External Function	monk_scripts/common/db_rcv.dsc
External Connection Establishment Function	db- stdver-conn-estab
External Connection Verification Function	db-stdver-conn-ver
External Connection Shutdown Function	db-stdver-conn-shutdown
Positive Acknowledgment Function	db-stdver-pos-ack
Negative Acknowledgment Function	db-stdver-neg-ack
Shutdown command notification function	db-stdver-shutdown
Database Setup (Fill in the information for these fields. See **"Database Setup" on page 33** for details.)	Database Type: Database name: User name: Encrypted Password:

9 Save this file and exit from the Settings page to return to the e*Gate Enterprise GUI.

4.2.4 Create the Collaboration Rules

The next step is creating the Collaboration Rules that will be associated with the Event Types in this schema. There are two Collaboration Rules for this sample schema, as follows:

- db_rcv

- no_xlate

Details for creating and configuring both are in the sections that follow.

db_rcv

This Collaboration Rule is associated with *db_rcv_struct* for output Event Types, and *db_rcv_in* for input Event Types. To create *db_rcv,* do the following:

1 Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2 In the Navigator, select the **Collaboration Rules** folder.

3 On the Palette, click

4 Enter *db_rcv* as the name of the new Collaboration Rule, then click **OK.**

5 Select *db_rcv*, then click to edit its properties.

6 In the **Service** field on the **General** tab, select the **Copy** as the Collaboration Service.

7 On the **Subscriptions** tab, select *db_rcv_in* as the required input Event Type.

8 On the **Publications** tab, select *db_struct* as the default output Event Type.

9 Click on **OK**.

no_xlate

This Collaboration Rule is associated with *FileInEvent* Event Type for input, and the *db_rcv_in* Event Type for output. To create *no_xlate*, do the following:

1 Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2 In the Navigator, select the **Collaboration Rules** folder.

3 On the Palette, click

4 Enter *no_xlate* as the name of the new Collaboration Rule, then click **OK.**

5 Select *no_xlate*, then click to edit its properties.

6 In the **Service** field on the **General** tab, select **Pass Through** as the Collaboration Service.

7 On the **Subscriptions** tab, select *FileInEvent* as the required input Event Type.

8 On the **Publications** tab, select *db_rcv_in* as the default output Event Type.

9 Click on **OK**.

4.2.5 Create the Intelligent Queue

This sample schema uses one Intelligent Queue (IQ), which receives the *db_rcv_in* Event Type, and forwards it to the *dart_rcv* e*Way. To create this IQ, do the following:

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the IQ.

3 Open the Control Broker.

4 Select the IQ Manager.

5   On the Palette, click [icon] .

6   Enter *IQ1* as the name of the new IQ, then click **OK.**

7   Select *IQ1*, then click [icon] to edit its properties.

8   On the **General** Tab, select *SeeBeyond Standard* as the Service. The default *Event Type Get Interval* of 100 Milliseconds is satisfactory.

9   On the **Advanced** tab, make sure that *Simple publish/subscribe* is checked under the **IQ behavior** section.

10   Click **OK**.

4.2.6 Create the Collaborations

The final steps entail creating two collaborations and assigning them to the e*Ways. These Collaborations are:

- Pub

- Sub

The following sections provide the details for creating and modifying these Collaborations.

Pub

This Collaboration is a member of the *FileIn* e*Way, and applies the *no_xlate* Collaboration Rule. To create this Collaboration, do the following:

1   In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2   Open the host on which you want to create the Collaboration.

3   Select the Control Broker.

4   Select the *FileIn* e*Way to assign the Collaboration.

5   On the Palette, click [icon] .

6   Enter *Pub* as the name of the new Collaboration, then click **OK.**

7   Select *Pub*, then click [icon] to edit its properties.

8   From the **Collaboration Rules** list, select *no_xlate*.

9   In the **Subscriptions** area, click **Add** to define the input Event Type and source to which this Collaboration will subscribe.

A   From the **Event Type** list, select the *FileInEvent* Event Type.

B   From the **Source** list, select <EXTERNAL>.

10 In the **Publications** area, click **Add** to define the output Event Type and destination to which this Collaboration will publish.

  A From the **Event Types** list, select the *db_rcv_in* Event Type.

  B From the **Destination** list, select the *IQ1* Intelligent Queue.

11 Click on **OK**.

Sub

This Collaboration will be a member of the *dart_rcv* e*Way, and applies the *db_rcv Collaboration* Rule. To create this Collaboration, do the following

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select the Control Broker.

4 Select the *Dart_rcv* e*Way to assign the Collaboration.

5 On the Palette, click .

6 Enter *Sub* as the name of the new Collaboration, then click **OK.**

7 Select *Sub*, then click to edit its properties.

8 From the **Collaboration Rules** list, select *db_rcv*.

9 In the **Subscriptions** area, click **Add** to define the input Event Type and source to which this Collaboration will subscribe.

  A From the **Event Type** list, select the *db_rcv_in* Event Type.

  B From the **Source** list, select *Pub*.

10 In the **Publications** area, click **Add** to define the output Event Type and destination to which this Collaboration will publish.

  A From the **Event Types** list, select the *db_rcv_struct* Event Type.

  B From the **Destination** list, select <External>.

11 Click on **OK**.

4.2.7 Execute the Schema

To execute the *Sybase_Test* schema, do the following:

1 Go to the command line prompt, and enter the following:

```
stccb -rh hostname -rs Sybase_Test -un username -up user password
-ln hostname_cb
```

Substitute *hostname*, *username* and *user password* as appropriate.

2 Exit from the command line prompt, and start the e*Gate Monitor GUI.

3 When prompted, specify the hostname which contains the Control Broker you started in step 1 above.

4 Select the *Sybase_Test* schema.

5 After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, *hostname_*igmgr, then click on the right button of the mouse, and select **Start.**

6 Highlight each of the e*Ways, right click the mouse, and select **Start**.

4.3 Sample Two – Schedule Driven Database Access

This section presents a complete worked example of accessing the database on a schedule.

In this example, you will see how to set up

- The **Exchange Data with External** function which queries the database and sends the data to the Collaboration
- The Collaboration which passes the data onto the FileOut e*Way.
- The File e*Way which receives data from the Collaboration and writes it to a file.

To execute this example, you must be familiar with the e*Gate GUI so that you can:

- Create Event Type Definitions
- Create Collaborations
- Create and configure e*Ways

4.3.1 Overview

You will create a schema call "DatabasePoll". In this schema, you will create the following e*Ways, Event Types, Collaborations and Collaboration Rules:

e*Ways:

- DBPoll
- FileOut

Event Types:

- db_poll
- db_poll_struct

Collaborations

Collaboration Rules

4.3.2 Create and Configure e*Ways

Create a new schema called "DatabasePoll." In this schema, create these two e*Ways

- DBPoll
- FileOut

You will configure these e*Ways in the following sections.

Configuring the "FileOut" e*Way

For the FileOut e*Way, select "stcewfile.exe" for the executable file.

Edit or create a new configuration parameter set and call it "FileOut.cfg". Give the configuration parameters the values shown in the following table. Note that several parameters specify "Default" as their value. You do not need to change these when configuration the FileOut e*Way.

Table 6   Configuration Parameters for FileOut e*Way

Parameter Name	Value
General settings	
AllowingIncoming	NO
AllowingOutgoing	YES
Performance Testing	**Default**
Poller (inbound) settings:	**Default**
Outbound (send) settings:	
OutputDirectory	c:\egate\output (output file folder)
OutputFileName	Default
MultipleRecordsPerFile	Default
MaxRecordsPerFile	Default
Add EOL	Default
Performance Testing	Default

Configuring the "DBPoll" e*Way

For the "DBPoll" e*Way, select "stcgenericmonk.exe" for the executable file.

Edit or create a new configuration set and call it "DBPoll.cfg". Select "DART" when choosing the e*Way template.

Fill in the following configuration parameters according to **Table 7 on page 62**

Notice that the default values are correct for the General settings. For the Database settings, you must enter values that match the installed database that you wish to test against. Finally note that you specify the Monk function db_poll.dsc, but that this function is not created yet. You will create it in a later section.

Table 7   Configuration Parameters for DBPoll

Parameter Name	Value
Communications Settings	
Start Exchange Data Schedule	Repeatedly, every 1 hour
Stop Exchange Data Schedule	None
Exchange Data Interval	10
Down Timeout	Default
Up Timeout	Default
Resend timeout	Default
Zero Wait between successful Exchanges	No

Table 7 Configuration Parameters for DBPoll

Parameter Name	Value
Monk Configuration Settings	
Additional Path	<leave blank>
Auxiliary Library Directories	monk_library\dart
Monk Environment Initialization File	db-stdver-init
Startup Function	db- stdver-startup
Process Outgoing Message function	<leave blank>
Exchange Data With External function	monk_scripts/common/db_poll.dsc
External Connection Establishment function	db- stdver-conn-estab
External Connection Verification function	db-stdver-conn-ver
External Connection Shutdown function	db-stdver-conn-shutdown
Positive Acknowledgment function	db-stdver-pos-ack
Negative Acknowledgment Fuction	db-stdver-neg-ack
Shutdown command notification function	db-stdver-shutdown
Database Settings	
Type	Choose ORACLE8, or ORACLE8i
Database name	defined by developer
User name	testuser
Encrypted Password	testuser

4.3.3 Create Event Type Definitions

Create two Event Types:

Event Type Name	Event Type Definition File
"db_poll_out"	db_poll_out.ssc
"db_poll_struct"	db_poll_struct.ssc

4.3.4 Create Collaboration Rules

Create two Collaboration Rules with these specifications:

Collaboration Rule Name	db_poll
Service	Pass Through.
Subscription	db_poll_struct
Publication	db_poll_out
Collaboration Rule Name	**xlate**
Service	Monk.
Subscription	db_poll_out

Publication	db_poll_struct
Collaboration Rules Script	\monk_scripts\common\xlateput.tsc

4.3.5 Create the Queue

Create one queue named queue1. Set the iq_service to STC_standard.

4.3.6 Create the Collaboration

Create two collaborations, one for DatPoll and one for FileOut.

Collaboration Name	dart_poll_coll
Collaboration Rule	db_poll
Subscriptions	db_poll_struct (event type)
	EXTERNAL (publication)
Publications	db_poll_out (event type)
	q1 (destination)

Collaboration Name	FileOut_coll
Collaboration Rule	xlate
Subscriptions	db_poll_out (event type)
	Dart_poll_coll (source)
Publications	db_poll_struct (event type)
	EXTERNAL(destination)

4.3.7 Create Monk functions

Create the Monk function "db_poll.dsc" as follows:

```
(load "db_poll_struct.ssc")
(load "db_poll_out.ssc")
(define dart_poll
  (let ((input ($make-event-map db_poll_struct-delm
                                db_poll_struct-struct))
        (output ($make-event-map db_poll_out-delm
                                 db_poll_out-struct))
       )
    (lambda ()
      ($event-clear output)
      (begin
       (if (db-struct-select connection-handle
              ~input%db_poll_struct.db_employee "EMP_NO > 10")
         (begin
          (let ((fetch_stat ""))
           (do ((i 0 (+ i 1))) ((or (boolean? fetch_stat)))
             (set! fetch_stat (db-struct-fetch connection-handle
                                ~input%db_poll_struct.db_employee))
            (if (not (boolean? fetch_stat))
             (begin
              (copy-strip
```

```
                    ~input%db_poll_struct.db_employee.EMP_NO.VALUE
                    ~output%db_poll_out.db_employee[<i>].number "")
                 (copy-strip
                    ~input%db_poll_struct.db_employee.LAST_NAME.VALUE
                    ~output%db_poll_out.db_employee[<i>].last_name "")
                 (copy-strip
                    ~input%db_poll_struct.db_employee.FIRST_NAME.VALUE
                    ~output%db_poll_out.db_employee[<i>].first_name "")
                 (copy-strip
                    ~input%db_poll_struct.db_employee.RATE.VALUE
                    ~output%db_poll_out.db_employee[<i>].rate "")
                 (copy-strip
                    ~input%db_poll_struct.db_employee.LAST_UPDATE.VALUE
                    ~output%db_poll_out.db_employee[<i>].date "")
                 (display "Fetched ")
                 (display ~output%db_poll_out.db_employee[<i>])
                 (newline )
                )
                (begin
                )
              )
            )
            (if (eq? fetch_stat #f)
             (begin
              (display "db-struct-fetch failed: ")
              (display (db-get-error-str connection-handle))
              (newline )
              (if (db-check-connect)
               (begin
               )
               (begin
               )
              )
             )
             (begin
             )
            )
           )
          )
          (begin
           (display "db-struct-select failed: ")
           (display (db-get-error-str connection-handle))
           (newline )
          )
         )
        )
        (let ((result ($event->string output)))
          ($event-clear input)
          ($event-clear output)
          result)
   )))
```

4.4 Sample Monk Scripts

These sample scripts demonstrate how use the Monk functions. The samples work together to:

1   Initialize the Monk extensions.

2   Define and Bind Stored procedures.

3 Call Stored Procedures.

4 Login to a Database.

5 Insert, Update, Select, and Delete records in a database using dynamic SQL Statements.

6 Insert a binary image file into a database.

7 Retrieve an image from a database.

Details for the functions used in the samples can be found in **"Database Access Functions" on page 105**

4.4.1 Initializing Monk Extensions

The sample script shows how to initialize the Monk extensions.

```
(define EGATE "/eGate/client")

; routine to load DART Monk extension
(define (load-library extension)
    (define filename (string-append EGATE "/bin/" extension))
    (if (file-exists? filename)
        (load-extension filename)
        (begin
            (display (string-append "File " filename " does not
exist.\n"))
            (abort filename)
        )
    )
)

(load-library "stc_monkext.dll")

;;
;; define database variables, data source, user ID, and password
;;

(define database "SYBASE")

(load-library "stc_dbmonkext.dll")

(define dsn "database")
(define uid "Administrator")
(define pwd (encrypt-password uid "password"))
```

4.4.2 Supporting Functions for Sample Scripts

This sample script displays and defines values and parameters for stored procedures. For more details about functions used in this script, see **"Stored Procedure Functions" on page 149**

```
;;
;; stored procedure auxiliary functions
;;

; display parameter properties of the stored procedure
(define (display-proc-parameter-property hdbc hstmt prm-count)
    (display "parameter count = ") (display prm-count) (newline)
    (do ((i 0 (+ i 1))) ((= i prm-count))
```

```
                (display "parameter ")
                (display (db-proc-param-name hdbc hstmt i))
                (display ": type = ")
                (display (db-proc-param-type hdbc hstmt i))
                (display ", io = ")
                (display (db-proc-param-io hdbc hstmt i))
                (newline)
         )
    )

    ; display value of output parameters from stored procedure
    (define (display-proc-parameter-output-value hdbc hstmt prm-count)
       (do ((i 0 (+ i 1))) ((= i prm-count))
           (if (not (equal? (db-proc-param-io hdbc hstmt i) "IN"))
               (begin
                   (display "output parameter ")
                   (display (db-proc-param-name hdbc hstmt i))
                   (display " = ")
                   (display (db-proc-param-value hdbc hstmt i))
                   (newline)
               )
           )
       )
    )

    ; display column properties of the return result set
    (define (display-proc-column-property hdbc hstmt col-count)
       (display "column count = ") (display col-count) (newline)
       (do ((i 0 (+ i 1))) ((= i col-count))
           (display "column ")
           (display (db-proc-column-name hdbc hstmt i))
           (display ": type = ")
           (display (db-proc-column-type hdbc hstmt i))
           (newline)
       )
       (newline)
    )

    ; display column value of the return result set of the stored
    procedure
    (define (display-proc-column-value hdbc hstmt col-count)
       (define (fetch-next)
           (let ((result (db-proc-fetch hdbc hstmt)))
               (if (boolean? result)
                   result
                   (begin (display result) (newline) (fetch-next))
               )
           )
       )
       (fetch-next)
       (newline)
    )

    ; bind stored procedure and display parameter properties
    (define (bind-procedure hdbc proc)
       (let ((hstmt (db-proc-bind hdbc proc)))
           (if (statement-handle? hstmt)
               (begin
                   (display (string-append "bind stored procedure : " proc
    "\n"))
                   (define prm-count (db-proc-param-count hdbc hstmt))
                   (display-proc-parameter-property hdbc hstmt prm-count)
                   (newline)
                   (if (db-proc-return-exist hdbc hstmt)
```

```
                (begin
                   (display "return: type = ")
                   (display (db-proc-return-type  hdbc hstmt))
                   (newline)
                )
             )
             (newline)
          )
           (display (db-get-error-str hdbc))
        )
      hstmt
    )
)




;;
;; dynamic statement auxiliary functions
;;

; display parameter properties of the SQL statement
(define (display-stmt-parameter-property hdbc hstmt prm-count)
   (display "parameter count = ") (display prm-count) (newline)
   (do ((i 0 (+ i 1))) ((= i prm-count))
      (display "parameter #")
      (display i)
      (display ": type = ")
      (display (db-stmt-param-type hdbc hstmt i))
      (newline)
   )
   (newline)
)

; display column properties of the SQL statement
(define (display-stmt-column-property hdbc hstmt col-count)
   (display "column count = ") (display col-count) (newline)
   (do ((i 0 (+ i 1))) ((= i col-count))
      (display "column ")
      (display (db-stmt-column-name hdbc hstmt i))
      (display ": type = ")
      (display (db-stmt-column-type hdbc hstmt i))
      (newline)
   )
   (newline)
)

; display column value of the return result set of the SQL statement
(define (display-stmt-column-value hdbc hstmt)
   (define (fetch-next)
      (let ((result (db-stmt-fetch hdbc hstmt)))
         (if (boolean? result)
            result
            (begin (display result) (newline) (fetch-next))
         )
      )
   )
   (fetch-next)
   (newline)
)

; display row count affected by the execution of the SQL statement
(define (display-stmt-row-count hdbc hstmt)
   (let ((row-count (db-stmt-row-count hdbc hstmt)))
```

```
        (cond
           ((= row-count 0) (display "\n(no row affected)\n"))
           ((= row-count 1) (display "\n(1 row affected)\n"))
           (else (display (string-append "\n(" (number->string row-
count) " rows affected)\n")))
        )
     )
  )

  ; bind dynamic statement and display paramters and column properties
  (define (bind-statement hdbc stmt)
     (let ((hstmt (db-stmt-bind hdbc stmt)))
        (display (string-append "\nDynamic statement : " stmt "\n"))
        (if (statement-handle? hstmt)
           (begin
              (define prm-count (db-stmt-param-count hdbc hstmt))
              (display-stmt-parameter-property hdbc hstmt prm-count)

              (define col-count (db-stmt-column-count hdbc hstmt))
              (display-stmt-column-property hdbc hstmt col-count)
           )
           (display (db-get-error-str hdbc))
        )
        hstmt
     )
  )

  ; bind dynamic statement to input/output raw binary data
  (define (bind-binary-statement hdbc stmt)
     (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
        (display (string-append "\nDynamic statement : " stmt "\n"))
        (if (statement-handle? hstmt)
           (begin
              (define prm-count (db-stmt-param-count hdbc hstmt))
              (display-stmt-parameter-property hdbc hstmt prm-count)

              (define col-count (db-stmt-column-count hdbc hstmt))
              (display-stmt-column-property hdbc hstmt col-count)
           )
           (display (db-get-error-str hdbc))
        )
        hstmt
     )
  )
```

4.4.3 Logging In

This scripts provides a sample of a login script. For details about functions in this script, see **"General Connection Functions" on page 106**.

```
; define eGate path
(define EGATE "/eGate/client")

; load Monk basic extension
(define MONKLIB (string-append EGATE "/bin/stc_monkext.dll"))
(load-extension MONKLIB)

; load Monk database extension
(define database "SYBASE")
(define DARTLIB (string-append EGATE "/bin/stc_dbmonkext.dll"))
(load-extension DARTLIB)

; define data source, user ID, and password
```

```
(define dsn "database")
(define uid "Administrator")
(define pwd (encrypt-password uid "password"))

(define hdbc (make-connection-handle))
(display (string-append "\nDART Login " dsn " ...\n"))
(if (db-login hdbc dsn uid pwd)
   (begin
      (display "database login succeed !\n")
      (db-logout hdbc)
   )
   (display (db-get-error-str hdbc))
)
```

4.4.4 Calling Stored Procedures

This script gives an example of calling Stored Procedures. See **"Stored Procedure Functions" on page 149** for more details.

```
; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; call stored procedure and display results
(define (execute-procedure hdbc hstmt)
   (let ((prm-count (db-proc-param-count hdbc hstmt)))
      (if (db-proc-execute hdbc hstmt)
         (begin
            (do ((col-count (db-proc-column-count hdbc hstmt) (db-
proc-column-count hdbc hstmt)))
                ((or (not (number? col-count)) (= col-count 0)))
               (display-proc-column-property hdbc hstmt col-count)
               (display-proc-column-value hdbc hstmt col-count)
            )
            (display-proc-parameter-output-value hdbc hstmt prm-count)
            (if (db-proc-return-exist hdbc hstmt)
               (begin
                  (display "return: value = ")
                  (display (db-proc-return-value hdbc hstmt))
                  (newline)
               )
            )
         )
         (display (db-get-error-str hdbc))
      )
   )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
   (display (db-get-error-str hdbc))
)

(if (db-login hdbc dsn uid pwd)
   (begin
      (display "database login succeed !\n")

      ; bind the stored procedures
      (define hstmt1 (bind-procedure hdbc "pubs2.dbo.titleid_proc"))
      (define hstmt2 (bind-procedure hdbc "pubs2.dbo.history_proc"))

      ; call the stored procedure if the binding is successful
```

```
(display "\nGet information of book with title_id = PC8888\n")
(if (statement-handle? hstmt1)
    (begin
        (display "call stored procedure titleid_proc ...\n\n")
        (if (db-proc-param-assign hdbc hstmt1 0 "PC8888")
            (execute-procedure hdbc hstmt1)
            (display (db-get-error-str hdbc))
        )
    )
    (display (db-get-error-str hdbc))
)

(display "\nGet sales history of store with stor_id = 8042\n")
(if (statement-handle? hstmt2)
    (begin
        (display "call stored procedure history_proc ...\n\n")
        (if (db-proc-param-assign hdbc hstmt2 0 "8042")
            (execute-procedure hdbc hstmt2)
            (display (db-get-error-str hdbc))
        )
    )
    (display (db-get-error-str hdbc))
)

(if (not (db-logout hdbc))
    (display (db-get-error-str hdbc))
)
)
(display (db-get-error-str hdbc))
)
```

4.4.5 Using Dynamic SQL Statements

At this point in the sample, a user has connected to the Database and has successfully logged in. This section will show how to use Dynamic SQL statements to do the following:

- Insert Records
- Update Records
- Select Records
- Delete Records

For more details about the functions used in the following sample scripts see, **"Dynamic SQL Functions" on page 133**.

Inserting Records with Dynamic SQL Statements

```
; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
    (if (db-stmt-execute hdbc hstmt)
        (begin
            (display-stmt-row-count hdbc hstmt)
            #t
        )
        #f
```

```
        )
    )

    ; make new connection handle
    (define hdbc (make-connection-handle))
    (if (not (connection-handle? hdbc))
        (display (db-get-error-str hdbc))
    )

    (define stmt1 "insert into publishers values (?, ?, ?, ?)")
    (define stmt2 "insert into titleauthor select au_id, 'no_id', null,
null from authors where au_id not in (select au_id from titleauthor)")

    (if (db-login hdbc dsn uid pwd)
        (begin
            (display "\ndatabase login succeed !\n")
            ; change to pubs2 database
            (if (db-sql-execute hdbc "use pubs2")
                (begin
                    ; bind the dynamic statements
                    (define hstmt1 (bind-statement hdbc stmt1))
                    (define hstmt2 (bind-statement hdbc stmt2))

                    ; assign parameter and execute the dynamic statement
                    (if (statement-handle? hstmt1)
                        (begin
                            (display "\nInsert record pub_id = 1756 into
publishers table ...\n")
                            (if
                                (and
                                    (db-stmt-param-assign hdbc hstmt1 0 "1756")
                                  (db-stmt-param-assign hdbc hstmt1 1 "The Heath
Center")
                                    (db-stmt-param-assign hdbc hstmt1 2 "Oakland")
                                    (db-stmt-param-assign hdbc hstmt1 3 "CA")
                                )
                                (if (execute-statement hdbc hstmt1)
                                    (begin
                                        (display "\nCommit the insertion ...\n")
                                        (if (not (db-commit hdbc))
                                            (display (db-get-error-str hdbc))
                                        )
                                    )
                                    (display (db-get-error-str hdbc))
                                )
                                (display (db-get-error-str hdbc))
                            )
                        )
                    )

                    (if (statement-handle? hstmt2)
                        (begin
                            (display "\nInsert records into titleauthor table
...\n")
                            (if (execute-statement hdbc hstmt2)
                                (begin
                                    (display "\nCommit the insertion ...\n")
                                    (if (not (db-commit hdbc))
                                        (display (db-get-error-str hdbc))
                                    )
                                )
                                (display (db-get-error-str hdbc))
                            )
                        )
```

```
            )
          )
          (display (db-get-error-str hdbc))
        )
        (if (not (db-logout hdbc))
            (display (db-get-error-str hdbc))
        )
      )
      (display (db-get-error-str hdbc))
  )
```

Updating Records with Dynamic SQL Statements

```
; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
    (if (db-stmt-execute hdbc hstmt)
        (begin
            (display-stmt-row-count hdbc hstmt)
            #t
        )
        #f
    )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)

(define stmt1 "update publishers set city = ?, state = ? where pub_id
= ?")
(define stmt2 "update titleauthor set title_id = titles.title_id from
titleauthor, titles, authors where titles.title = ? and authors.au_id
= titleauthor.au_id and au_lname = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
        (display "\ndatabase login succeed !\n")
        ; change to pubs2 database
        (if (db-sql-execute hdbc "use pubs2")
            (begin
                ; bind the dynamic statements
                (define hstmt1 (bind-statement hdbc stmt1))
                (define hstmt2 (bind-statement hdbc stmt2))

                ; assign parameter and execute the dynamic statement
                (if (statement-handle? hstmt1)
                    (begin
                        (display "\nUpdate record pub_id = 1756 into
publishers table ...\n")
                        (if (and
                               (db-stmt-param-assign hdbc hstmt1 0 "Atlanta")
                               (db-stmt-param-assign hdbc hstmt1 1 "GA")
                               (db-stmt-param-assign hdbc hstmt1 2 "1756")
                            )
                            (if (execute-statement hdbc hstmt1)
                                (begin
                                    (display "\nCommit the update ...\n")
                                    (if (not (db-commit hdbc))
```

```
                                (display (db-get-error-str hdbc))
                            )
                        )
                        (display (db-get-error-str hdbc))
                    )
                    (display (db-get-error-str hdbc))
                )
            )
        )

        (if (statement-handle? hstmt2)
            (begin
                (display "\nUpdate records in titleauthor table
...\n")
                (if (and
                        (db-stmt-param-assign hdbc hstmt2 0 "The
Psychology of Computer Cooking")
                        (db-stmt-param-assign hdbc hstmt2 1 "Stringer")
                    )
                    (if (execute-statement hdbc hstmt2)
                        (begin
                            (display "\nCommit the updates ...\n")
                            (if (not (db-commit hdbc))
                                (display (db-get-error-str hdbc))
                            )
                        )
                        (display (db-get-error-str hdbc))
                    )
                    (display (db-get-error-str hdbc))
                )
            )
        )
        (display (db-get-error-str hdbc))
    )
    (if (not (db-logout hdbc))
        (display (db-get-error-str hdbc))
    )
)
(display (db-get-error-str hdbc))
)
```

Selecting Records with Dynamic SQL Statements

```
; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

; execute dynamic statement and display results
(define (execute-statement hdbc hstmt)
    (if (db-stmt-execute hdbc hstmt)
        (begin
            (display-stmt-column-value hdbc hstmt)
            #t
        )
        #f
    )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
```

```
            (display (db-get-error-str hdbc))
    )

    (define stmt1 "select au_fname, au_lname, phone from authors where
    city = ?")
    (define stmt2 "select distinct title, price from titles, salesdetail
    where titles.title_id = salesdetail.title_id and discount = ?")

    (if (db-login hdbc dsn uid pwd)
        (begin
            (display "\ndatabase login succeed !\n")
            ; change to pubs2 database
            (if (db-sql-execute hdbc "use pubs2")
                (begin
                    ; bind the dynamic statements
                    (define hstmt1 (bind-statement hdbc stmt1))
                    (define hstmt2 (bind-statement hdbc stmt2))

                    ; assign parameter and execute the dynamic statement
                    (if (statement-handle? hstmt1)
                        (begin
                            (display "\nList author in Oakland ...\n\n")
                            (if (db-stmt-param-assign hdbc hstmt1 0 "Oakland")
                                (execute-statement hdbc hstmt1)
                                (display (db-get-error-str hdbc))
                            )
                            (display "\nList author in Salt Lake City ...\n\n")
                            (if (db-stmt-param-assign hdbc hstmt1 0 "Salt Lake
    City")
                                (execute-statement hdbc hstmt1)
                                (display (db-get-error-str hdbc))
                            )
                        )
                    )

                    (if (statement-handle? hstmt2)
                        (begin
                            (display "\nList book title and price with 40 percent
    discount ...\n\n")
                            (if (db-stmt-param-assign hdbc hstmt2 0 "40")
                                (execute-statement hdbc hstmt2)
                                (display (db-get-error-str hdbc))
                            )
                        )
                        (display (db-get-error-str hdbc))
                    )
                )
                (display (db-get-error-str hdbc))
            )
            (if (not (db-logout hdbc))
                (display (db-get-error-str hdbc))
            )
        )
        (display (db-get-error-str hdbc))
    )
```

Deleting Records with Dynamic SQL Statements

```
    ; load Monk database extension
    (load "demo-init.monk")
    (load "demo-common.monk")

    ; execute dynamic statement and display results
    (define (execute-statement hdbc hstmt)
```

```
    (if (db-stmt-execute hdbc hstmt)
        (begin
            (display-stmt-row-count hdbc hstmt)
            #t
        )
        #f
    )
)

; make new connection handle
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
    (display (db-get-error-str hdbc))
)

(define stmt1 "delete from publishers where pub_id = ?")
(define stmt2 "delete from titleauthor where title_id = ? or au_ord =
null")

(if (db-login hdbc dsn uid pwd)
    (begin
        (display "\ndatabase login succeed !\n")
        ; change to pubs2 database
        (if (db-sql-execute hdbc "use pubs2")
            (begin
                ; bind the dynamic statements
                (define hstmt1 (bind-statement hdbc stmt1))
                (define hstmt2 (bind-statement hdbc stmt2))

                ; assign parameter and execute the dynamic statement
                (if (statement-handle? hstmt1)
                    (begin
                        (display "\nDelete record with pub_id = 1756 from
publishers table ...\n")
                        (if (db-stmt-param-assign hdbc hstmt1 0 "1756")
                            (if (execute-statement hdbc hstmt1)
                                (begin
                                    (display "\nCommit the deletion ...\n")
                                    (if (not (db-commit hdbc))
                                        (display (db-get-error-str hdbc))
                                    )
                                )
                                (display (db-get-error-str hdbc))
                            )
                            (display (db-get-error-str hdbc))
                        )
                    )
                )

                (if (statement-handle? hstmt2)
                    (begin
                        (display "\nDelete records from titleauthor table
...\n")
                        (if (db-stmt-param-assign hdbc hstmt2 0 "no_id")
                            (if (execute-statement hdbc hstmt2)
                                (begin
                                    (display "\nCommit the deletion ...\n")
                                    (if (not (db-commit hdbc))
                                        (display (db-get-error-str hdbc))
                                    )
                                )
                                (display (db-get-error-str hdbc))
                            )
                            (display (db-get-error-str hdbc))
```

```
                    )
                )
            )
        )
        (display (db-get-error-str hdbc))
    )
    (if (not (db-logout hdbc))
        (display (db-get-error-str hdbc))
    )
)
(display (db-get-error-str hdbc))
)
```

4.4.6 Inserting a Binary Image to a Database

This sample shows how to insert a Binary Image into a Database. It uses both Static and Dynamic SQL functions. See **"Database Access Functions" on page 105** for more details.

```
; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")
(load-library "stc_monkutils.dll")

(define (query-exist hdbc hstmt id)
    (let ((rec-count 0) (result '#()))
        (if (db-stmt-param-assign hdbc hstmt 0 id)
            (if (db-stmt-execute hdbc hstmt)
                (begin
                    (set! result (vector-ref (db-stmt-fetch hdbc hstmt) 0))
                    (set! rec-count (string->number result))
                    (set! result (db-stmt-fetch-cancel hdbc hstmt))
                    (if (> rec-count 0)
                        (begin
                            (display "author image already exist\n")
                            #t
                        )
                        #f
                    )
                )
                (begin
                    (display (db-get-error-str hdbc))
                    #f
                )
            )
            (begin
                (display (db-get-error-str hdbc))
                #f
            )
        )
    )
)

(define (execute-statement hdbc hstmt)
    (let ((col-count (db-stmt-column-count hdbc hstmt)) (row-count 0))
        (if (db-stmt-execute hdbc hstmt)
            (begin
                (if (> col-count 0)
                    (if (not (stmt-display-column-value hdbc hstmt col-
count))
                        (display (db-get-error-str hdbc))
                    )
```

```
            )
            (set! row-count (db-stmt-row-count hdbc hstmt))
            (if (boolean? row-count)
                (display (db-get-error-str hdbc))
                (display (string-append "number of image insert = "
(number->string row-count) "\n"))
            )
            (newline)
            #t
        )
        #f
    )
  )
)

(define author-id "756-30-7391")
(define image-file "webbie.jpg")
(define image-type "JPEG")
(define image-width "1024")
(define image-height "768")

(define image-port (open-input-file image-file))
(define image (read image-port 1048576))
(close-port image-port)
(define bytesize (number->string (string-length image)))
(define image-hex (string->hexdump image))

(define hdbc (make-connection-handle))
(display (connection-handle? hdbc)) (newline)

(define stmt0 "select count(0) from au_pix where au_id = ?")
(define stmt1 "insert into au_pix (au_id, format_type, bytesize,
pixwidth_hor, pixwidth_vert) values (?, ?, ?, ?, ?)")
(define stmt2 (string-append "update au_pix set pic = 0x00 where au_id
= '" author-id "'"))
(define stmt3 (string-append "declare @val varbinary(16)\nselect @val
= textptr(pic) from au_pix where au_id = '" author-id "'\nwritetext
au_pix.pic @val 0x" image-hex))
(define stmt4 "delete from au_pix where au_id = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
        (display "\ndatabase login succeed\n")
        (display (db-dbms hdbc)) (newline)
        (display (db-std-timestamp-format hdbc)) (newline)
        (display (db-sql-execute hdbc "use pubs2")) (newline)
        (define hquery  (bind-statement hdbc stmt0))
        (define hinsert (bind-statement hdbc stmt1))
        (define hdelete (bind-statement hdbc stmt4))

        (if (and
                (statement-handle? hquery)
                (statement-handle? hdelete)
            )
            (if (query-exist hdbc hquery author-id)
                (begin
                    (display "delete existing author image record\n")
                    (if (db-stmt-param-assign hdbc hdelete 0 author-id)
                        (if (db-stmt-execute hdbc hdelete)
                            (db-commit hdbc)
                            (begin
                                (display (db-get-error-str hdbc))
                                (abort "failed to delete existing record")
                            )
```

```
                )
                (display (db-get-error-str hdbc))
            )
        )
    )
)

(if (statement-handle? hinsert)
    (begin
        (display "insert author image record\n")
        (if (and
                (db-stmt-param-assign hdbc hinsert 0 author-id)
                (db-stmt-param-assign hdbc hinsert 1 image-type)
                (db-stmt-param-assign hdbc hinsert 2 bytesize)
                (db-stmt-param-assign hdbc hinsert 3 image-width)
                (db-stmt-param-assign hdbc hinsert 4 image-height)
            )
            (if (execute-statement hdbc hinsert)
                (begin
                    (display "update author image data\n")
                    (if (db-sql-execute hdbc stmt2)
                        (if (db-sql-execute hdbc stmt3)
                            (db-commit hdbc)
                            (begin
                                (display (db-get-error-str hdbc))
                                (db-rollback hdbc)
                            )
                        )
                        (display (db-get-error-str hdbc))
                    )
                )
                (display (db-get-error-str hdbc))
            )
            (display (db-get-error-str hdbc))
        )
    )
)

(if (not (db-logout hdbc))
    (display (db-get-error-str hdbc))
)
)
(display (db-get-error-str hdbc))
)
```

4.4.7 Retrieving an Image from a Database

This sample shows how to Retrieve an image from a Database. It uses both Static and Dynamic SQL functions. See **"Database Access Functions" on page 105** for more details.

```
; load Monk database extension
(load "demo-init.monk")
(load "demo-common.monk")

(define (get-image hdbc hstmt)
    (do (
            (result (db-stmt-fetch hdbc hstmt) (db-stmt-fetch hdbc
hstmt))
            (first_name "")
            (file_type "")
            (file_name "")
```

```
          (width "")
          (height "")
          (output_port '())
       )
      ((boolean? result) result)
     (set! first_name (vector-ref result 0))
     (set! file_type (strip-trailing-whitespace (vector-ref result
1)))
     (set! width (strip-trailing-whitespace (vector-ref result 2)))
     (set! height (strip-trailing-whitespace (vector-ref result 3)))
     (cond
        ((string=? file_type "JPEG") (set! file_name (string-append
first_name ".jpg")))
        ((string=? file_type "GIF") (set! file_name (string-append
first_name ".gif")))
        ((string=? file_type "PICT") (set! file_name (string-append
first_name ".pct")))
        ((string=? file_type "TIF") (set! file_name (string-append
first_name ".tif")))
        ((string=? file_type "Sun raster") (set! file_name (string-
append first_name ".ras")))
        (else (set! file_name (string-append first_name ".raw")))
     )
     (if (file-exists? file_name)
        (file-delete file_name)
     )
     (display (string-append "picture name = " file_name "\n"))
     (display (string-append "picture size = " width " x " height
"\n\n"))
     (set! output_port (open-output-file file_name))
     (display (vector-ref result 4) output_port)
     (close-port output_port)
   )
)

(define (execute-statement hdbc hstmt)
   (let ((col-count (db-stmt-column-count hdbc hstmt)) (row-count 0))
     (if (db-stmt-execute hdbc hstmt)
        (begin
           (if (> col-count 0)
              (if (not (get-image hdbc hstmt))
                 (display (db-get-error-str hdbc))
              )
           )
           (set! row-count (db-stmt-row-count hdbc hstmt))
           (if (boolean? row-count)
              (display (db-get-error-str hdbc))
              (display (string-append "number of image retrieved = "
(number->string row-count) "\n"))
           )
           (newline)
           #t
        )
        #f
     )
   )
)

(define hdbc (make-connection-handle))
(display (connection-handle? hdbc)) (newline)

(define author-id "756-30-7391")
```

```
(define stmt "select au_fname, format_type, pixwidth_hor,
pixwidth_vert, pic from authors, au_pix where authors.au_id =
au_pix.au_id and au_pix.au_id = ?")

(if (db-login hdbc dsn uid pwd)
    (begin
        (display "\ndatabase login succeed !\n")

        ; change to pubs2 database
        (if (db-sql-execute hdbc "use pubs2")
            (begin
                ; bind the select statement
                (define hselect (bind-binary-statement hdbc stmt))

                ; execute the dynamic statement
                (if (statement-handle? hselect)
                    (begin
                        (display "select author's picture\n")
                        (if (db-stmt-param-assign hdbc hselect 0 author-id)
                            (if (not (execute-statement hdbc hselect))
                                (display (db-get-error-str hdbc))
                            )
                            (display (db-get-error-str hdbc))
                        )
                    )
                )
            )
            (display (db-get-error-str hdbc))
        )

        (if (not (db-logout hdbc))
            (display (db-get-error-str hdbc))
        )
    )
    (display (db-get-error-str hdbc))
)
```

Sybase e*Way Functions

The functions described in this chapter control the Sybase e*Way's basic operations as well as those needed for database access.

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

- **Standard e*Way Functions** on page 82
- **Generic e*Way Built-in Functions** on page 97
- **Database Access Functions** on page 105

Specific examples and detailed sample scripts are provided in **"Sample Monk Scripts" on page 65**.

5.1 Standard e*Way Functions

The functions described in this section can only be used by the functions defined within the e*Way's configuration file. None of the functions are available to Collaboration Rules scripts executed by the e*Way. The current suite of Standard e*Way functions included are:

db-stdver-init on page 83

db-stdver-startup on page 84

db-stdver-conn-estab on page 85

db-stdver-conn-ver on page 87

db-stdver-conn-shutdown on page 88

db-stdver-pos-ack on page 89

db-stdver-neg-ack on page 90

db-stdver-shutdown on page 91

db-stdver-proc-outgoing on page 92

db-stdver-proc-outgoing-stub on page 94

db-stdver-data-exchg on page 96

db-stdver-data-exchg-stub on page 97

db-stdver-init

Syntax

```
(db-stdver-init)
```

Description

db-stdver-init begins the initialization process for the e*Way. The function loads all monk extension libraries used by other e*Way functions.

Parameters

None.

Return Values

string

If a **FAILURE** string is returned, the e*Way shutdowns. Any other return indicates success.

Throws

None.

db-stdver-startup

Syntax

```
(db-stdver-startup)
```

Description

db-stdver-startup is used for instance specific function loads and **"env"** setup.

Parameters

None.

Return Values

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

Throws

None.

Examples

```
(define db-stdver-startup
 (lambda ( )
 (let ((result "SUCCESS"))
 (display "[++] Executing e*Way external startup function.")
 result
 )
))
```

db-stdver-conn-estab

Syntax

```
(db-stdver-conn-estab)
```

Description

db-stdver-conn-estab is used to establish external system connection.This function will

- construct a new connection handle
- call db-long to connect to database
- set up timestamp format if required
- set up maximum long data buffer limit if required
- bind dynamic SQL statement and stored procedures.

Parameters

None.

Return Values

string
UP or **SUCCESS** if connection established, anything else if connection not established.

Throws

None.

Additional Information

To use standard database time format, add the following function call to this function: (db-std-timestamp-format *connection-handle*) after the (db-bind) call.

For "Maximum Long Data Size" the library allocates an internal buffer for each SQL_LONGVARCHAR and SQL_LONGVARBINARY data, when the SQL statement or stored procedure that contains these datatypes are bound. The default size of each internal data buffer is 1024K(1048576) bytes. If the user needs to handle long data larger than this default value, add the following function call to specify the maximum data size:

```
(db-max-long-data-size connection-handle maximum-data-size)
```

See **db-max-long-data-size** on page 115 for more information.

Examples

```
(define db-stdver-conn-estab
 (lambda ( )
 (let ((result "DOWN")(last_dberr ""))
 (display "[++] Executing e*Way external connection establishment
function.")
 (display "db-stdver-conn-estab: logging into the database with:\n")
 (display "DATABASE NAME = ")
 (display DATABASE_SETUP_DATABASE_NAME)
 (newline )
 (display "USER NAME = ")
 (display DATABASE_SETUP_USER_NAME)
 (newline )
 (set! connection-handle (make-connection-handle))
```

```
   (if (connection-handle? connection-handle)
   (begin
    (if (db-login connection-handle DATABASE_SETUP_DATABASE_NAME
DATABASE_SETUP_USER_NAME DATABASE_SETUP_ENCRYPTED_PASSWORD)
    (begin
       (db-bind )
     (set! result "UP")
    )
    (begin
     (set! last_dberr (db-get-error-str connection-handle))
     (display last_dberr)
     (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_CANTCONN"
"ALERTINFO_FATAL" "0" "Cannot connect to database" (string-append
"Failed to connect to database: " DATABASE_SETUP_DATABASE_NAME "with
error" last_dberr) 0 (list))
     (newline )
     (db-logout connection-handle)
     (set! result "DOWN")
    )
    )
   )
   (begin
    (set! result "DOWN")
    (display "Failed to create connection handle.")
    (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_UNUSABLE"
"ALERTINFO_FATAL" "0" "database connection handle creation error"
"Failed to create database connection handle" 0 (list))
   )
   )
   result
   )
))
```

db-stdver-conn-ver

Syntax

```
(db-stdver-conn-ver)
```

Description

db-stdver-conn-ver is used to verify whether external system connection is established.

Parameters

None.

Return Values

string
UP or **SUCCESS** if connection established, anything else if connection not established.

Throws

None.

Additional Information

To use standard database time format, add the following function call to this function:
(db-std-timestamp-format *connection-handle*) after the (db-bind) call.

Examples

```
(define db-stdver-conn-ver
 (lambda ( )
 (let ((result "DOWN")(last_dberr ""))
  (display "[++] Executing e*Way external connection verification
function.")
  (display "db-stdver-conn-ver: checking connection status...\n")
  (cond ((string=? STCDB "SYBASE") (db-sql-select connection-handle
"verify" "select getdate()")) ((string=? STCDB "ORACLE8i") (db-sql-
select connection-handle "verify" "select sysdate from dual"))
((string=? STCDB "ORACLE8") (db-sql-select connection-handle "verify"
"select sysdate from dual")) ((string=? STCDB "ORACLE7") (db-sql-
select connection-handle "verify" "select sysdate from dual")) (else
(db-sql-select connection-handle "verify" "select {fn NOW()}")))
  (if (db-alive connection-handle)
  (begin
   (db-sql-fetch-cancel connection-handle "verify")
   (set! result "UP")
  )
  (begin
   (set! last_dberr (db-get-error-str connection-handle))
   (display last_dberr)
   (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_LOSTCONN"
"ALERTINFO_FATAL" "0" "Lost connection to database" (string-append
"Lost connection to database: " DATABASE_SETUP_DATABASE_NAME "with
error" last_dberr) 0 (list))
   (set! result "DOWN")
  )
  )
 result
 )
))
```

db-stdver-conn-shutdown

Syntax

```
(db-stdver-conn-shutdown string)
```

Description

db-stdver-conn-shutdown is called by the system to request that the interface disconnect from the external system, preparing for a suspend/reload cycle. Any return value indicates that the suspend can occur immediately, and the interface is placed in the down state.

Parameters

Name	Type	Description
string	string	When the e*Way calls this function, it passes the string "SUSPEND_NOTIFICATION" as the parameter.

Return Values

string

Any return indicates that the external is ready to suspend. The user may choose to define the return string as **SUCCESS**, **#t** (true), or simply and empty string.

Throws

None.

Examples

```
(define db-stdver-conn-shutdown
 (lambda ( message-string )
 (let ((result "SUCCESS"))
 (comment "Std e*Way connection shutdown function" "[++] Usage:
Function called by system to request that the interface disconnect
from the external system, preparing for a suspend/reload cycle. Any
return value indicates that the suspend can occur immediately, and the
interface is placed in the down state. [++] Input to expect: Function
should not expect input. [++] Expected return values: anything
indicates that the external is ready to suspend.n")
 (comment "db-stdver-conn-shutdown [++] Implementation specific
comment" "none")
 (display "[++] Executing e*Way external connection shutdown
function.")
 (display message-string)
 (db-logout connection-handle)
 result
 )
))
```

db-stdver-pos-ack

Syntax

```
(db-stdver-pos-ack message-string)
```

Description

db-stdver-pos-ack is used to send a positive acknowledgment to the external system, and for post processing after successfully sending data to e*Gate.

Parameters

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

Return Values

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

CONNERR indicates a loss of connection with the external, client moves to a down state and attempts to connect, on reconnect **pos-ack** function is re executed.

Throws

None.

Examples

```
(define db-stdver-pos-ack
 (lambda ( message-string )
 (let ((result ""))
 (display "[++] Executing e*Way external positive acknowledgement
function.")
 (display message-string)
 result
 )
))
```

db-stdver-neg-ack

Syntax

```
(db-stdver-neg-ack message-string)
```

Description

db-stdver-neg-ack is used to send a negative acknowledgment to the external system, and for post processing after failing to send data to e*Gate.

Parameters

Name	Description
message-string	The Event for which a negative acknowledgment is sent.

Return Values

string
An empty string indicates a successful operation.

CONNERR indicates a loss of connection with the external, client moves to a down state and attempts to connect, on reconnect **neg-ack** function is re-executed.

Throws

None.

Examples

```
(define db-stdver-neg-ack
 (lambda ( message-string )
 (let ((result ""))
 ( (display "[++] Executing e*Way external negative acknowledgement
function.")
 (display message-string)
 result
 )
))
```

db-stdver-shutdown

Syntax

```
(db-stdver-shutdown shutdown_notification)
```

Description

db-stdver-shutdown is called by the system to request that the external shutdown, a return value of SUCCESS indicates that the shutdown can occur immediately, any other return value indicates that the shutdown Event must be delayed. The user is then required to execute a **shutdown-request** call from within a monk function to allow the requested shutdown process to continue.

Parameters

Name	Type	Description
shutdown_notification	string	When the e*Way calls this function, it passes the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

string
SUCCESS allows an immediate shutdown to occur, anything else delays shutdown until a **shutdown-request** is executed successfully**.**

Throws

None.

Examples

```
(define db-stdver-shutdown
 (lambda ( message-string )
 (let ((result "SUCCESS"))
 (display "[++] Executing e*Way external shutdown command
notification function.")
 result
 )
))
```

db-stdver-proc-outgoing

Syntax

```
(db-stdver-proc-outgoing message-string)
```

Description

db-stdver-proc-outgoing is used for sending a received message (Event) from e*Gate to the external system.

Parameters

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

Return Values

string
An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way compares the number of attempts made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- If the number of attempts does not exceed the maximum, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, increments the "resend attempts" counter for that message, then repeats the attempt to send the message.

- If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.

CONNERR indicates that there is a problem communicating with the external system. First, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way calls the **External Connection Establishment function** according to the **Down Timeout** schedule, and rolls back the message (Event) to the IQ from which it was obtained.

DATAERR indicates that there is a problem with the message (Event) data itself. First, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way increments its "failed message (Event)" counter, and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see **Journal File Name** on page 15), the message (Event) is journaled.

If a string other than the following is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function.

Throws

None.

Examples

```
(define db-stdver-proc-outgoing
```

```
(lambda ( message-string )
(let ((result ""))
(display "[++] Executing e*Way external process outgoing message
function.")
 (display message-string)
 result
 )
))
```

db-stdver-proc-outgoing-stub

Syntax

```
(db-stdver-proc-outgoing-stub message-string)
```

Description

db-stdver-proc-outgoing-stub is used as a place holder for the function entry point for sending an Event received from e*Gate to the external system. When the interface is configured as an inbound only connection, this function should not be used. This function is used to catch configuration problems.

Parameters

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

Return Values

string
An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way compares the number of attempts it has made to send the Event to the number specified in the Max Resends per Messages parameter, and does one of the following:

- If the number of attempts does not exceed the maximum, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter, increments the "resend attempts" counter for that message, then repeats the attempt to send the message.

- If the number of attempts exceeds the maximum, the function returns false and rolls back the message to the e*Gate IQ from which it was obtained.

CONNERR indicates that there is a problem communicating with the external system. First, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way calls the **External Connection Establishment function** according to the **Down Timeout** schedule, and rolls back the message (Event) to the IQ from which it was obtained.

DATAERR indicates that there is a problem with the message (Event) data itself. First, the e*Way pauses the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way increments its "failed message (Event)" counter, and rolls back the message (Event) to the IQ from which it was obtained. If the e*Way's journal is enabled (see **Journal File Name** on page 15) the message (Event) is journaled.

If a string other than the following is returned, the e*Way creates an entry in the log file indicating that an attempt has been made to access an unsupported function.

Throws

None.

Examples

```
(define db-stdver-proc-outgoing-stub
 (lambda ( message-string )
 (let ((result ""))
 (display "[++] Executing e*Way external process outgoing message
function stub.")
 (display message-string)
 (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_INTEREST"
"ALERTINFO_NONE" "0" "Possible configuration error." (string-append
"Default eway process outgoing msg function passed following message:
" msg) 0 (list))
 result
 )
))
```

db-stdver-data-exchg

Syntax

```
(db-stdver-data-exchg)
```

Description

db-stdver-data-exchg is used for sending a received Event from the external system to e*Gate. The function expects no input.

Parameters

None.

Return Values

string

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

A message-string indicates successful operation and the Event is sent to e*Gate.

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect, on reconnect this function is re-executed with the same input message**.**

Throws

None.

Examples

```
(define db-stdver-data-exchg
 (lambda ( )
 (let ((result ""))
 (display "[++] Executing e*Way external data exchange function.")
 result
 )
))
```

db-stdver-data-exchg-stub

Syntax

```
(db-stdver-data-exchg-stub)
```

Description

db-stdver-data-exchg-stub is used as a place holder for the function entry point for sending an Event from the external system to e*Gate. When the interface is configured as an outbound only connection, this function should not be called. The function expects no input.

Parameters

None.

Return Values

string

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

A message-string indicates a successful operation and the Event is sent to e*Gate.

CONNERR indicates the loss of connection with the external, and the client moves to a down state. Any attempts to connect, or reconnect are re-executed with the same input message.

Throws

None.

Examples

```
(define db-stdver-data-exchg-stub
 (lambda ( )
 (let ((result ""))
 (event-send "ALERTCAT_OPERATIONAL" "ALERTSUBCAT_INTEREST"
"ALERTINFO_NONE" "0" "Possible configuration error." "Default eway
data exchange function called." 0 (list))
 result
 )
))
```

5.2 Generic e*Way Built-in Functions

The Generic e*Way Functions are general purpose functions available to most e*Ways. They are available to Collaboration Rules scripts executed by the e*Way. The functions in this category control the user-defined operations. The Built-in functions are:

start-schedule

Syntax

```
(start-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

stop-schedule

Syntax

```
(stop-schedule)
```

Description

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-down

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

get-logical-name

Syntax

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

Description

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

Parameters

None.

Return Values

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

Throws

None.

event-send-to-egate

Syntax

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

Description

event-send-to-egate sends an Event from the e*Way. If the external collaboration(s) is successful in publishing the Event to the outbound queue, the function returns **#t** (true); otherwise **#f (**false) is returned.

Parameters

Name	Type	Description
string	string	The data to be sent to the e*Gate system

Return Values

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

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.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown-request 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** (see **"Shutdown Command Notification Function" on page 33**). Once this function is called, shutdown proceeds immediately.

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

Parameters

None.

Return Values

None.

Throws

None.

5.3 Database Access Functions

The Database Access Functions permit data to be manipulated using SQL insert, update and delete statements as well as procedure calls within the connected database. These functions are available to Collaboration Rules scripts executed by the e*Way.

These Monk functions are used to interface with the application database, and are divided into five categories, as follows:

- **General Connection Functions** on page 106
- **Static SQL Functions** on page 121:
- **Dynamic SQL Functions** on page 133:
- **Stored Procedure Functions** on page 149:
- **Message Event Functions** on page 179

5.3.1 General Connection Functions

The functions provide for basic operations such as logging in/out of the database, connecting to the database, and generating error messages.

make-connection-handle

Syntax

```
(make-connection-handle)
```

Description

This function constructs the *connection handle*.

Parameters

None.

Return Values

Boolean
Returns **#t** (true) if a Monk object of type connection-handle is returned; otherwise, returns **#f** (false) if the function fails to create a connection-handle. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hdbc (make-connection-handle))
```

Explanation

The above example creates a connection handle variable called hdbc.

connection-handle?

Syntax

```
(connection-handle? any-variable)
```

Description

connection-handle? determines whether or not the input argument is a *connection-handle* datatype.

Parameters

Name	Type	Description
any-variable	variable	A single variable of any data type is required.

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
(define hdbc (make-connection-handle))
(if (not (connection-handle? hdbc))
  (display (db-get-error-str hdbc))
)
```

Explanation

The above example creates a connection handle called hdbc. An error message is displayed if the newly defined hdbc is not a connection handle.

db-login

Syntax

```
(db-login connection-handle data-source user-name password)
```

Description

db-login allocates the resources and performs login to a database system.

db-login requires an encrypted password. If a password was specified in the Database Setup section of the e*Way Editor, it has already been encrypted. (See **"Database Setup" on page 33**.)

If a password was defined within a Monk function (which is not encrypted), you must use the monk function **encrypt-password** found in the e*Gate Monk extension library stc_monkext.dll:

```
encrypt-password encryption key  plain password
```

where encryption key is public knowledge, i.e., in this case user id, and plain password is the password to be encrypted.

The encrypt-password API returns an encrypted password string to be used with db-login.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
data-source	string	The name of the data source.
user-name	string	The database user login name.
password	string	The database user login password.

Note: The data_source, user_name, and password must not be an empty string.

Return Values

Boolean
Returns **#t** when the end of the fetch cycle is reached; otherwise, returns **#f**. Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
...
(define hdbc (make-connection-handle) )
(define uid "James")
(define pwd (encrypt-password uid "12345"))
(if (db-login hdbc "Payroll" "James" "12345")
  ...
)
```

Explanation

The above example shows how to use the connection handle (hdbc) to log into the data source "Payroll" as "James" with the password "12345."

db-logout

Syntax

```
(db-logout connection-handle)
```

Description

db-logout performs a disconnect from the database system and releases the connection handle resources.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(define hdbc (make-connection handle) )
(if (db-login hdbc "dsn" "uid" "pwd")
  ...
  (db-logout hdbc)
)
...
```

Explanation

The above example shows how to disconnect from a database. For every **db-login**, there should be a corresponding **db-logout**.

Notes

Roll back or commit a transaction before you call **db-logout**. If a transaction is neither committed nor rolled back, it is automatically rolled back before logout.

db-alive

Syntax

```
(db-alive connection-handle)
```

Description

db-alive is used to determine if the cause of a failing Sybase e*Way operation is due to a broken connection. It returns whether or not the database connection was alive during the last call to any Sybase e*Way procedure that sends commands to the database server.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean

Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc "dsn" "uid" "pwd")
  (begin
    (define sql_statement "select * from person where sex = 'M'")
    (do ((status #t)) ((not status))
      (if (db-sql-select hdbc "male" sql_statement)
        (begin
          ...
        )
        (begin
          (display (db-get-error-str hdbc))
          (set! status (db-alive hdbc))
        )
      )
    )
    (display "lost database connection !\n"))
    (db-logout hdbc))
  )
)
```

Explanation

The example above illustrates an application that is looking for a certain record in the "person" table of the "Payroll" database. The function exits the loop only if it loses the connection to the database.

Notes

- Most procedures can detect a dead connection handle except **db-commit** and **db-rollback**. Therefore, when the procedure returns false, users must check for loss of connection.

- Once the **db-alive** returns **#f** (false) to indicate either a dead connection handle or an un-available database server, all the subsequent Sybase e*Way function calls associated with that connection handle will not be executed, with the exception of **db-logout**. Each of these procedures returns false with a "lost database connection" error message.

- Once determined the connection handle is not alive, the only course of action the user can take is to log out from that connection handle, redefine a new connection handle, and try to reconnect to the database.

db-std-timestamp-format

Syntax

```
(db-std-timestamp-format connection-handle)
```

Description

db-std-timestamp-format sets the date to SQL92 standard format--"YYYY-MM-DD HH:MI:SS.SSS"--at the connection level and must be called immediately after login.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Notes

- When the user logs into the database, the database server has a default timestamp format set. The default format can be any non-standard format.

- The db-std-timestamp-format API forces the input and output of the timestamp format to the standard SQL92 standard format. Using standard format frees the user from reformatting each time data is exchanged with other applications.

db-max-long-data-size

Syntax

```
(db-max-long-data-size connection-handle size)
```

Description

db-max-long-data specifies the maximum buffer size for the long data allowing the data to be larger than one megabyte. Long data may have a range in size up to two gigabytes ($2x10^9$). In order to limit the memory consumption of the library, it is necessary to use this function to specify the maximum data size expected. Long data larger than the specified size is truncated. This data size is used for buffer allocation for both long data columns as well as long data parameters.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
size	integer	This parameter is used to identify the buffer size of the specified long datatype. Note: The default buffer size is one megabyte.

Return Values

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Additional Information

The default maximum buffer size for long data type is one megabyte (1048576). It is not necessary to call this function unless the long data is in excess of one megabyte.

db-commit

Syntax

```
(db-commit connection-handle)
```

Description

db-commit performs all transactions specified by the connection.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if
  (and
    (db-sql-execute hdbc "delete from employee where first_name =
'John'")
    (db-sql-execute hdbc "update employee set first_name = 'Mary'
where ssn = 123456789")
  )
  (db-commit hdbc)
  (begin
    (display (db-get-error-str hdbc))
    (db-rollback hdbc)
  )
)
...
```

Explanation

This example shows that if the application can successfully delete "John's record" and update "Mary's record" it commits the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

db-rollback

Syntax

```
(db-rollback connection-handle)
```

Description

db-rollback rolls back the entire transaction for the connection.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if
  (and
    (db-sql-execute hdbc "delete from employee where first_name =
'John'")
    (db-sql-execute hdbc "update employee set first_name = 'Mary'
where ssn = 123456789")
  )
  (db-commit hdbc)
  (begin
    (display (db-get-error-str hdbc))
    (newline)
    (db-rollback hdbc)
  )
)
...
```

Explanation

This example shows that if the application can successfully delete "John's record" and update "Mary's record," it commits the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

statement-handle?

Syntax

```
(statement-handle? any-variable)
```

Description

statement-handle? determines whether or not the input argument is a statement handle datatype.

Parameters

Name	Type	Description
any-variable	variable	A single variable of any datatype is required.

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
(define hstmt (db-proc-bind hdbc "test"))
(if (not (statement-handle? hstmt))
  (display (db-get-error-str hdbc))
)
```

Explanation

The above example creates a statement handle called hstmt, then it displays an error message if the newly defined hstmt is not a statement handle.

db-get-error-str

Syntax

```
(db-get-error-str connection-handle)
```

Description

db-get-error-str returns the last error message, and is used when a value of **#f** (false) is returned.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.

Return Values

string

A simple error message is returned. To parse the return error message when it contains an error, use the two standard files that define the error message structure and display the contents of each component of the error message.

```
SYBASE - sybmsg.ssc, sybmsg_display.monk
```

Throws

None.

Examples

```
...
(if (db-sql-execute hdbc "delete from employee where
first_name='John')
  (db-commit hdbc)
  (display (db-get-error-str hdbc))
)
...
```

Explanation

This example shows that if the application can successfully delete "John's record" it commits the transaction. Otherwise, the application prints out the error message and rolls back the same transaction. Each commit begins a new transaction automatically.

```
(if (db-login hdbc dsn uid pwd)
  (begin
   (display "database login succeed !\n")
   (if (db-sql-execute hdbc "INSERT INTO UNKNOWN VALUES (NULL)")
     (db-commit hdbc)
     (sybmsg-display (db-get-error-str hdbc))
   )
   (if (not (db-logout hdbc))
     (sybmsg-display (db-get-error-str hdbc))
   )
   )
   (sybmsg-display (db-get-error-str hdbc))
)
```

Output of the above example (db-get-error-str hdbc)

```
SERVER|208|16|42000|1||UNKNOWN not found. Specify owner.objectname or
use sp_help to check whether the object exists (sp_help may produce
lots of output). DART|63|STCDB_X_conn_sql_exec_len||unable to execute
SQL statement
```

Output of the above example (sybmsg-display (db-get-error-str hdbc)

```
SERVER message #0:
msg_source        : SERVER
msg_number        : 208
severity          : 16
sql_state         : 42000
line_number       : 1
proc_name         :
msg_string        : UNKNOWN not found. Specify owner.objectname or use
sp_help to check whether the object exists (sp_help may produce lots
of output).

DART message #0:
msg_source        : DART
msg_number        : 63
function          : STCDB_X_conn_sql_exec_len
msg_target        :
msg_string        : unable to execute SQL statement
```

5.3.2 Sybase SQL Type Support

The following table shows the supported SQL datatypes and the corresponding native datatype for the Sybase database.

Table 8   Sybase SQL Type Support

SQL Type Name	SQL Datatype	Sybase Datatype
SQL_BIT	BIT	bit
SQL_BINARY	BINARY (n)	binary (n)
SQL_VARBINARY	VARBINARY (n)	varbinary (n)
SQL_CHAR	CHAR (n)	char (n)
SQL_VARCHAR	VARCHAR (n)	varchar (n)
SQL_DECIMAL	DECIMAL (p, s)	decimal (p,s)
SQL_NUMERIC	NUMERIC (p, s)	numeric (p, s)
SQL_TINYINT	TINYINT	tinyint
SQL_BIGINT	BIGINT	N/A
SQL_SMALLINT	SMALLINT	smallint
SQL_INTEGER	INTEGER	int
SQL_REAL	REAL	real
SQL_FLOAT	FLOAT(p)	FLOAT(p)
SQL_DOUBLE	DOUBLE PRECISION	double precision
SQL_DATE	DATE	N/A
SQL_TIME	TIME	N/A
SQL_TIMESTAMP	TIMESTAMP	datetime

Table 8 Sybase SQL Type Support

SQL Type Name	SQL Datatype	Sybase Datatype
SQL_LONGVARCHAR	LONG VARCHAR	text
SQL_LONGVARBINARY	LONG VARBINARY	image

*Sybase float (p) specifies a floating point number with precision range from 1 to 126.

Note: *All variable precision datatypes require precision values.*

5.3.3 Static SQL Functions

These SQL functions represent statements that are embedded in the program source code.

- **db-sql-format** on page 122
- **db-sql-execute** on page 124
- **db-sql-select** on page 125
- **db-sql-fetch** on page 126
- **db-sql-fetch-cancel** on page 127
- **db-sql-column-names** on page 128
- **db-sql-column-types** on page 130
- **db-sql-column-values** on page 132

db-sql-format

Syntax

```
(db-sql-format data-string SQL-type)
```

Description

db-sql-format returns a formatted string of the *data-string*, so it can be used in an SQL statement as a literal value of a corresponding *SQL-type*.

In the current implementation, only the SQL_CHAR, SQL_VARCHAR, SQL_DATE, SQL_TIME, and SQL_TIMESTAMP SQL-types is formatted. If the <data-string> is an empty string, the procedure returns a NULL value for all SQL data types except SQL_CHAR and SQL_VARCHAR.

Parameters

Name	Type	Description
data-string	string	A data string to be used as a literal value in an SQL statement.
SQL-type	string	The SQL data type string, i.e., SQL_VARCHAR.

Return Values

string
Returns a formatted string used as a data value in an SQL statement.

Throws

None.

Examples

```
(define last-name (db-sql-format "O'Reilly" "SQL_VARCHAR"))
(define timestamp (db-sql-format "1998-02-19 12:34:56"
SQL_TIMESTAMP"))
(define sql-stmt (string-append "update employee set lastname =
"last-name ", MODIFYTIME = "timestamp "WHERE SSN = 123456789"))
(if (db-login hdbc "Payroll" "user" "password")
  (begin
    (if (db-sql-execute hdbc sql-stmt)
      (db-commit hdbc)
      (begin
        (display (db-get-error-str hdbc))
        (newline)
        (db-rollback hdbc)
      )
    )
    ...
    (db-logout hdbc)
  )
)
```

Explanation

The example above illustrates how the program uses **db-sql-format** to format the last name and the timestamp and use the results as part of an SQL statement.

Remarks

- For SQL_CHAR and SQL_VARCHAR (SQL datatypes) **db-sql-format** places a single quotation mark (') before and after the *data-string*, and expand each single quotation mark in the *data-string* to two single quotation mark characters.

- If you use the (timestamp) Monk built-in function to insert the timestamp to an Event Type Definition, you should specify the following format for it to be accepted by the **db-sql-format** function: "%Y-%m-%d %H:%M:%S"

- This function only works with the **db-std-timestamp-format** because the **db-sql-format** API handles only standard timestamp format.

The following table shows the typical <data-string> and the corresponding results of the formatting for an SQL statement.

Table 9   SQL Statement Formats

SQL_type Value	Data_string Value	Formatted Result String
SQL_CHAR	This is a string	'This is a string.'
SQL_VARCHAR	O'Reilly	'O' 'Reilly'
SQL_TIMESTAMP	1998-02-19 12:34:56.789	'1998-02-19 12:34:56.789'

db-sql-execute

Syntax

```
(db-sql-execute connection-handle SQL-stmt)
```

Description

db-sql-execute carries out the specified SQL statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
SQL-stmt	string	The SQL statement being executed

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false).
Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-login hdbc "Payroll" "James" "12345")
  (begin
    ...
    (if (db-sql-execute hdbc "insert into employee values('John'...)")
      (db-commit hdbc)
      )
      )
    (display (db-get-error-str hdbc))
)
...
```

Explanation

This example shows that if the application can successfully log into the data source
"Payroll," it inserts a record into the table "employee."

Notes

- Use the **db-sql-select** function to execute a select statement.

  Use this function to execute either a DDL (data definition language) statement, i.e.,
  create a table, alter a table, delete a table, or a DML (data manipulation language)
  statement, i.e., select a table, insert a value into a database, update a table.

  All DML transactions must be closed, or an error occurs, before using a DDL
  statement.

- Use **db-commit** or **db-rollback** to commit and roll back transactions.

db-sql-select

Syntax

```
(db-sql-select connection-handle selection-name SQL-statement)
```

Description

db-sql-select executes an SQL SELECT statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.
SQL-statement	string	The SELECT statement.

Return Values

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false).
Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-sql-select hdbc "GreaterThan25" "select * employee where age
> 25")
  (begin
    (display (db-sql-fetch hdbc "GreaterThan25"))
    (newline)
    (db-sql-fetch-cancel hdbc "GreaterThan25")
  )
  (display (db-get-error-str hdbc))
)
...
```

Explanation

This example shows that the application selects the first record of employees who are older than age 25, by fetching the records one at a time and cancelling the remainder of the return records.

db-sql-fetch

Syntax

```
(db-sql-fetch connection-handle selection-name)
```

Description

db-sql-fetch "fetches" the result of a SELECT statement. The statement handle is "free" after the function fetches the last record.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

string
Returns a comma, delimited string containing all the column values for the record.

Boolean
Returns **#t** (true) at the end of the "fetch cycle," when no more records are available to "fetch"; otherwise, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-sql-select hdbc "GreaterThan25" "select * employee where age
> 25")
  (begin
    (display (db-sql-fetch hdbc "GreaterThan25"))
    (newline)
    (db-sql-fetch-cancel hdbc "GreaterThan25")
  )
  (begin
    (display (db-get-error-str hdbc))
    (newline)
  )
)
...
```

Explanation

This example shows that the application selects the first record of employees who are older than age 25, by fetching the record once and cancelling the rest of the records.

Notes

The return result is temporarily stored in RAM. The buffer is allocated when **db-sql-select** is called. The maximum size of the buffer is determined by the operating system.

db-sql-fetch-cancel

Syntax

```
(db-sql-fetch-cancel connection-handle selection-name)
```

Description

db-sql-fetch-cancel closes the cursor associated with an SQL SELECT statement and cancels the fetch command. It also frees up the memory allocation for the selection.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-sql-select hdbc "GreaterThan25" "select * employee where age
> 25")
  (begin
    (define result (db-sql-fetch hdbc "GreaterThan25"))
    (if (not (boolean? result))
      (db-sql-fetch-cancel hdbc "GreaterThan25")
      (if (not result)
        (begin
          (display (db-get-error-str hdbc))
          (newline)
        )
      )
    )
  )
  (begin
    (display (db-get-error-str hdbc))
    (newline)
  )
)
...
```

Explanation

This example shows that the application selects the first record of employees who are older than age 25, by fetching the record once and cancelling the rest of the records.

db-sql-column-names

Syntax

```
(db-sql-column-names connection-handle selection-name)
```

Description

db-sql-column-names returns a vector of column names which are the result of an SQL
SELECT statement identified by the parameter selection-name. This procedure can be
called after a SQL SELECT statement has been issued successfully.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

string
This function returns a vector of column names in string format if successful.

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false).
Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title='manager'")
(if (db-login hdbc "dsn" "uid" "pwd")
  (begin
    (if (db-sql-select hdbc "manager" selection)
      (begin
        (define name-array (db-sql-column-names hdbc "manager"))
        (if (vector? name-array)
          (begin
            (display "name of the first column: ")
            (display (vector-ref name-array 0))
            (newline)
            ...
          )
          (begin
            (display (db-get-error-str hdbc))
            (newline)
          )
        )
        (display (db-get-error-str hdbc))
        (newline)
      )
    )
    (if (db-alive hdbc)
      (begin
        ...
```

```
            )
        )
        (db-logout hdbc)
    )
)
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program displays the name of the first column.

db-sql-column-types

Syntax

```
(db-sql-column-types connection-handle selection-name)
```

Description

db-sql-column-types returns a vector of column types which are the result of an SQL SELECT statement identified by the parameter selection-name. This procedure can be called after a SQL SELECT statement has been issued successfully. Refer to the description for **db-bind-proc** for a list of SQL-type names.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

string
This function returns a vector of column types in string format if successful.

Boolean
If the string type is unavailable for any reason, this function returns a **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title= 'manager'")
   (if (db-sql-select hdbc "manager" selection)
     (begin
       (define type-array (db-sql-column-types hdbc "manager"))
       (if (vector? type-array)
         (begin
           (display "type of the first column:")
           (display (vector-ref type-array 0))
           (newline)
           ...
           ...
         )
         (begin
           (display (db-get-error-str hdbc))
           (newline)
         )
       )
       (display (db-get-error-str hdbc))
     )
   )
   (if (db-alive hdbc)
     (begin
       ...
     )
   )
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program displays the first column type.

db-sql-column-values

Syntax

```
(db-sql-column-values connection-handle selection-name)
```

Description

db-sql-column-values returns a vector of column values, which is the result of an SQL FETCH statement identified by the parameter selection-name. This procedure can be called after a SQL FETCH statement has been issued successfully.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
selection-name	string	The name that identifies the selection.

Return Values

string
Returns a vector of SQL values in string format if successful.

Boolean
If the values string is unavailable for any reason, this function returns a **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title= 'manager'")
    (if (db-sql-select hdbc "manager" selection)
      (do ((result "") (value-array #())) ((boolean? result))
        (set! result (db-sql-fetch hdbc "manager"))
        (if (not (boolean? reslt))
          (begin
            (set! value-array (db-sql-column-values hdbc "manager"))
            (do ((index 0 (+ index 1)) (count (vector-length value-
array))
              ((= index count))
              (display (vector-ref value-array index))
              (display "\t")
            )
            (newline)
          )
          (if (not result) (display (db-get-error-str hdbc)))
        )
      )
      (begin
        (display (db-get-error-str hdbc))
        (newline)
      )
    )
    (if (db-alive hdbc)
      (begin
        ...
```

```
            )
        )
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program loops through a fetch cycle. Within each fetch loop, the program displays the value of each column in the same line, separated by a tab character.

Notes

- A successful **db-sql-fetch** call returns a string which contains the concatenation of all column values with the comma (,) character as the separator. Although this single string is suitable for display purposes, the user must parse the result string to retrieve the value of each column.

- If the value of the column contains the comma (,) character, the user is unable to differentiate the comma data from the comma separator. Therefore, **db-sql-column-values** returns the result as a vector of values in string type to allow the user to make use of the **vector-ref** function to retrieve the value of each column and avoid any parsing problem.

5.3.4 Dynamic SQL Functions

Dynamic SQL statements are built and executed at run time versus Static SQL statements that are embedded within the program source code. The dynamic SQL functions supported by the Sybase e*Way are:

- **db-stmt-bind** on page 138
- **db-stmt-bind-binary** on page 139
- **db-stmt-param-count** on page 140
- **db-stmt-param-type** on page 141
- **db-stmt-param-assign** on page 142
- **db-stmt-execute** on page 143
- **db-stmt-fetch** on page 144
- **db-stmt-fetch-cancel** on page 145
- **db-stmt-column-count** on page 146
- **db-stmt-column-name** on page 147
- **db-stmt-column-type** on page 148
- **db-stmt-row-count** on page 149

Dynamic statements do not require knowledge of the complete structure of an SQL statement before building the application. This allows for run time input to provide information about the database objects to query.

The application can be written so that it prompts the user or scans a file for information that is not available at compilation time.

In Dynamic statements, the four steps of processing an SQL statement take place at run time, but they are performed only once. Execution of the plan takes place only when EXECUTE is called. **Figure 26 on page 137** shows the difference between Dynamic SQL with immediate execution, and Dynamic SQL with prepared execution.

The dynamic statement functions have been developed to support Long Raw and/or Long data. Use of the Long Raw format, allows the buffer size to be increased to two gigabytes from the default of one megabyte for standard data.

Benefits of Dynamic SQL

Using dynamic SQL commands, an application can prepare a "generic" SQL statement once and execute it multiple time. Statements can also contain markers for parameter values to be supplied at execution time, so that the statement can be executed with varying inputs.

Limitations of Dynamic SQL

Dynamic SQL has some significant limitations. They are

1 Performance of Dynamic SQL Commands

A dynamic SQL implementation of an application generally performs worse than an implementation where permanent stored procedures are created and the client program invokes them with RPC (remote procedure call) commands.

When an stored procedure is created for an application, SQL statement compilation and optimization are performed once when the procedure is created. With a dynamic SQL application, compilation and optimization are performed every time the client program runs. A dynamic SQL implementation also incurs database space overhead because each instance of the client program must create separate compiled versions of the application's prepared statements. When you design an application to use stored procedures and RPC commands, all instances of the client program can share the same stored procedures.

2 SQL Server Restrictions and Database Requirements

SQL Server implements dynamic SQL using temporary stored procedures. A temporary stored procedure is created when an SQL statement is prepared, and destroyed when that prepared statement is deallocated.

As a result of this implementation, an application accessing SQL Server and using dynamic SQL is subject to the restrictions of SQL Server stored procedures. Some of the implications of this are:

   ◆ Temporary tables are destroyed when the prepared statement is deallocated.

   ◆ Parameters of text and image datatypes are not supported.

   ◆ The maximum number of parameters supported is 255.

   ◆ If the dynamic SQL statement itself executes a stored procedure (with a Transact-SQL execute statement), output parameter values and the return status are unavailable to the client application.

Note: *See the Transact-SQL User's Guide for a complete discussion of stored procedures.*

3 Retrieving a text or image column

Using ct_get_data to Fetch text and image Values

Only columns that follow the last column bound with the bind call are available for use with ct_get_data.

For example, if an application selects for columns, all of which are text, and binds the first and third columns to program variables, then the application cannot use ct_get_data to retrieve the text contained in the second column. However, it can use ct_get_data to retrieve the text in the fourth column. Applications that control the select statement can reorder the select list so that the text and image columns come at the end.

4 Updating a text or image Column

Text or image columns can be updated two ways:

Embed the new value in the text of language command that sends an update statement. The advantage of this method is simplicity. The disadvantage is that the application must send the entire value at once. This method may not be appropriate for very large columns (that is, larger than that for which the program can allocate). Note that SQL Server requires the value to be embedded in the command text, and not passed as a command parameter. SQL Server does not allow parameters of type text or image.

Figure 25   Dynamic Statement Functions Flow Chart
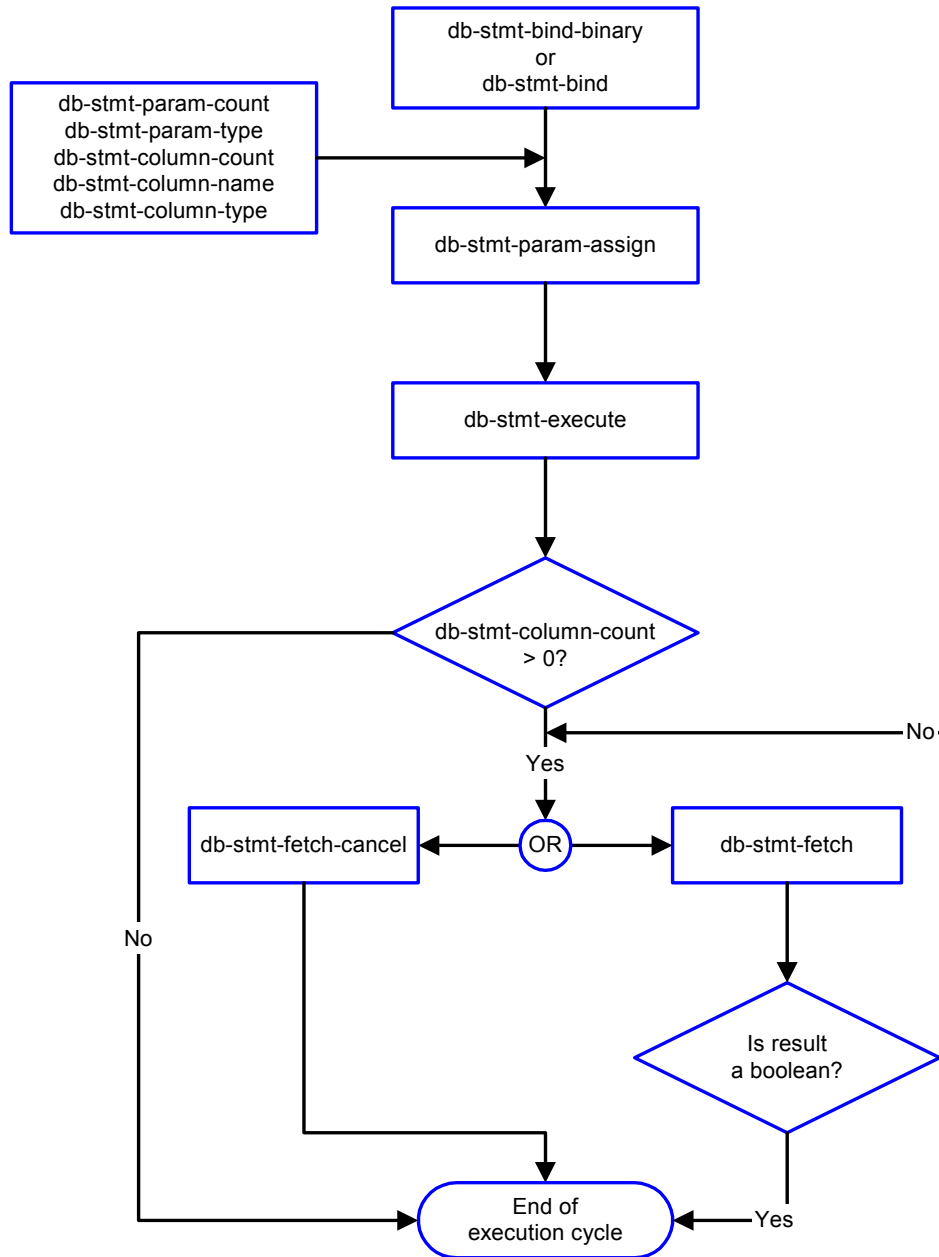
Figure 26   Example of Dynamic SQL processing

SQL Statement   **Dynamic SQL**

Select A,B,C From X, Y Where A<500 AND C = 'EFG'	db-sql-execute
Parse Statement	PREPARE statement Runtime
Validate Statement	db-stmt-bind
Optimize Statement	EXECUTE IMMEDIATE statement
Generate access plan	EXECUTE IMMEDIATE statement
Execute access plan	db-stmt-execute
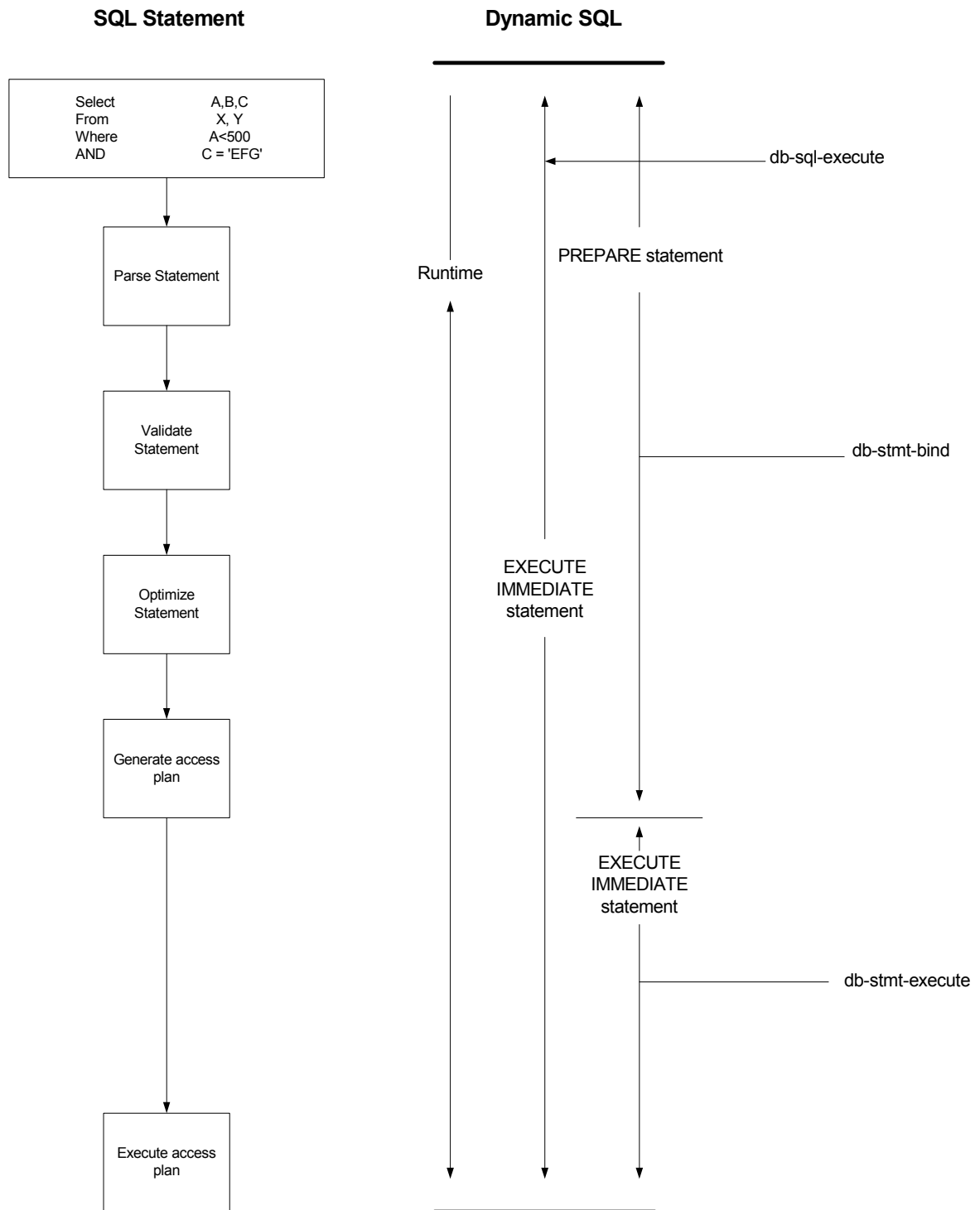
db-stmt-bind

Syntax

```
(db-stmt-bind connection-handle dynamic-SQL-statement)
```

Description

db-stmt-bind binds the dynamic statement specified. The binary data type requires input or output parameters in hexadecimal format.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
dynamic SQL-statement	string	The dynamic statement to be bound.

Return Values

statement handle
The statement handle that identifies the dynamic statement specified.

Boolean
If unsuccessful, returns **#f**. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Additional Information

- Use **db-stmt-bind-binary to** input/output binary data in the raw format.

- The long column should appear at the end of the selection list when selecting the long data type column.

db-stmt-bind-binary

Syntax

```
(db-stmt-bind-binary connection-handle dynamic-SQL-statement)
```

Description

db-stmt-bind-binary binds the dynamic statement specified. The binary data type is input and output with raw format.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
dynamic SQL-statement	string	The dynamic statement to be bound.

Return Values

statement handle
The statement handle that identifies the dynamic statement specified.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-param-count

Syntax

```
(db-stmt-param-count connection-handle statement-handle)
```

Description

db-stmt-param-count retrieves the number of parameters in the dynamic statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-proc-bind.

Return Values

integer
Returns a number, which represents the number of parameters for the dynamic statement specified, when successful.

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-param-type

Syntax

```
(db-stmt-param-type connection-handle statement-handle index)
```

Description

db-stmt-param-type retrieves the SQL datatype of the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-proc-bind.
index	integer	The number between 0 and db-stmt-param-count minus 1.

Return Values

string
   If successful, **db-stmt-param-type** returns a string which represents the SQL datatype.

Boolean
   If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

   None.

Additional Information

   The parameter datatype defaults to SQL-VARCHAR because the Sybase OCI API is unable to report datatypes for each bound parameter in a dynamic statement. All bound parameters default to the **VARCHAR** datatype. This allows Sybase to implicitly convert the data string of each parameter into the correct data value of the parameter at the execution of the dynamic statement.

db-stmt-param-assign

Syntax

```
(db-stmt-param-assign connection-handle statement-handle index value)
```

Description

db-stmt-param-assign assigns the parameter and executes the dynamic statement of a specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-proc-bind.
index	integer	The number between 0 and db-stmt-param-count minus 1.
value	string	The value to be assigned to the parameter.

Return Values

Boolean
If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-execute

Syntax

```
(db-stmt-execute connection-handle statement-handle statement-node)
```

Description

db-stmt-execute assigns the parameter and executes the dynamic statement of a specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-proc-bind.
statement-node	string	The Event path of a dynamic statement structure node generated by the stcstruct.exe program.

Return Values

Boolean

If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-fetch

Syntax

```
(db-stmt-fetch connection-handle statement-handle)
```

Description

db-stmt-fetch retrieves the column values of the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

A Vector and a Boolean
Returns a vector containing all the column values and at the end of the "fetch cycle" returns **#t** (true) when there are no more records available. If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

db-stmt-fetch-cancel

Syntax

```
(db-stmt-fetch-cancel connection-handle statement-handle)
```

Description

db-stmt-fetch-cancel terminates the current "fetch" cycle.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-column-count

Syntax

```
(db-stmt-column-count connection-handle statement-handle)
```

Description

db-stmt-column-count returns the number of columns in the return result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

number
Returns a number greater than zero (0) when the record set is available.

Boolean
If no record set is available, the return value is **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-column-name

Syntax

```
(db-stmt-column-name connection-handle statement-handle index)
```

Description

db-stmt-column-name returns the name string of the specified column in the result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified.
index	integer	An integer equal to -- 0 to db-stmt-column-count minus 1.

Return Values

string
Returns the name string if successful.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-column-type

Syntax

```
(db-stmt-column-type connection-handle statement-handle index)
```

Description

db-stmt-column-type retrieves the SQL datatype of the specified column in the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified.
index	integer	The SQL datatype of the specified column in the record set --0 to db-stmt-column-count minus 1.

Return Values

string
Returns a string of SQL datatype when successful.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-stmt-row-count

Syntax

```
(db-stmt-row-count connection-handle statement-handle index)
```

Description

db-stmt-column-size retrieves the number of rows affected by the execution of the SQL statement.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	string	The statement handle that identifies the dynamic statement specified.
index	integer	The SQL datatype of the specified column in the record set --0 to db-stmt-column-count minus 1.

Return Values

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

5.3.5 Stored Procedure Functions

These functions are executed only when called, and when specific criteria has been satisfied, such as successful login, and valid connection to the database.

db-proc-bind

Syntax

```
(db-proc-bind connection-handle procedure-name)
```

Description

db-proc-bind binds the input/output parameters of the stored procedure specified.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
procedure-name	string	The stored procedure to be bound.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

Scenario #1 — Sample code for db-proc-bind

```
(define hstmt (db-proc-bind hdbc "test")
(if (not (statement-handle? hstmt)
  (display "fail to bind stored procedure test\n")
)
```

Scenario #2 — Sample code for db-proc-bind for
Sybase DBMS only

```
(db-sql-execute hdbc "use sybsystemprocs")
(define hstmt (db-proc-bind hdbc "sp_help")
(if (not (statement-handle? hstmt)
  (display "fail to bind stored procedure\n")
)
(db-sql-execute hdbc "use my_db")
```

Explanation

For Sybase DBMS only: When you want to bind the Sybase system procedure, such as **sp_help** you must switch to the target database, in this case *sybsystemprocs*, before you bind any system procedure. When binding is completed, you can return to the original database.

db-proc-bind-binary

Syntax

```
(db-proc-bind-binary connection-handle dynamic-SQL-statement)
```

Description

db-proc-bind-binary binds the dynamic statement specified. The binary data type is input and output in binary format.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
dynamic SQL-statement	string	The dynamic statement to be bound.

Return Values

statement handle
The statement handle that identifies the dynamic statement specified.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

db-proc-param-count

Syntax

```
(db-proc-param-count connection-handle statement-handle)
```

Description

db-proc-param-count retrieves the number of parameters in the stored procedure.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

number
Returns a number, which represents the number of parameters for the stored procedure specified, when successful.

Boolean
If the number is unavailable due to a problem within one of the arguments, the function returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
    (display "database login succeed !\n")
    (define hstmt (db-proc-bind hdbc "TEST_PROC"))
    (if (statement-handle? hstmt)
      (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
        (display "parameter ")
         (display (db-proc-param-name hdbc hstmt i))
        (display ": type = ")
        (display (db-proc-param-type hdbc hstmt i))
        (display ", io = ")
        (display (db-proc-param-io hdbc hstmt i))
        (newline)
      )
      (display (db-get-error-str hdbc))
    )
    ...
    ...
  )
  (display (db-get-error-str hdbc))
)
```

Notes

The PL/SQL table type parameter is treated as a column rather than a parameter because it contains multiple values; a parameter contains only one value. Subsequently, the return value of this function is the number of non-table type parameters only. The **db-proc-column-count** function returns the number of table type parameters.

db-proc-param-name

Syntax

```
(db-proc-param-name connection-handle statement-handle param-index)
```

Description

db-proc-param-name retrieves the name of the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

string
Returns the string containing the name of the parameter.

Boolean
Returns **#f** (false) if unable to return the string containing the name of the parameter.
Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
    (display "database login succeed !\n")
    (define hstmt (db-proc-bind hdbc "TEST_PROC"))
    (if (statement-handle? hstmt)
      (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
        (display "parameter ")
        (display (db-proc-param-name hdbc hstmt i))
        (display ": type = ")
        (display (db-proc-param-type hdbc hstmt i))
        (display ", io = ")
        (display (db-proc-param-io hdbc hstmt i))
        (newline)
        )
      (display (db-get-error-str hdbc))
      )
    ...
    ...
    )
  (display (db-get-error-str hdbc))
  )
```

db-proc-param-type

Syntax

```
(db-proc-param-type connection-handle statement-handle param-index)
```

Description

db-proc-param-type retrieves the SQL datatype of the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

string
If successful, **db-proc-param-type** returns a string which represents the SQL datatype.

Boolean
If an error occurred, returns **#f** (false). Use **db-get-error-str** to obtain the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
    (display "database login succeed !\n")
    (define hstmt (db-proc-bind hdbc "TEST_PROC"))
    (if (statement-handle? hstmt)
      (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
        (display "parameter ")
        (display (db-proc-param-name hdbc hstmt i))
        (display ": type = ")
        (display (db-proc-param-type hdbc hstmt i))
        (display ", io = ")
        (display (db-proc-param-io hdbc hstmt i))
        (newline)
      )
      (display (db-get-error-str hdbc))
    )
    ...
    ...
  )
  (display (db-get-error-str hdbc))
)
```

db-proc-param-io

Syntax

```
(db-proc-param-io connection-handle statement-handle param-index)
```

Description

db-proc-param-io retrieves the IO type for the specified parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

string
Returns an IO type string as **IN**, **OUT**, or **INOUT**

Boolean
Otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
    (display "database login succeed !\n")
    (define hstmt (db-proc-bind hdbc "TEST_PROC"))
    (if (statement-handle? hstmt)
      (do ((i 0 (+ i 1))) ((= i (db-proc-param-count hdbc hstmt)))
        (display "parameter ")
        (display (db-proc-param-name hdbc hstmt i))
        (display ": type = ")
        (display (db-proc-param-type hdbc hstmt i))
        (display ", io = ")
        (display (db-proc-param-io hdbc hstmt i))
        (newline)
      )
      (display (db-get-error-str hdbc))
    )
    ...
    ...
  )
  (display (db-get-error-str hdbc))
)
```

db-proc-param-assign

Syntax

```
(db-proc-param-assign connection-handle statement-handle param-index
param-value)
```

Description

This function assigns the value of an IN or INOUT parameter and places that value into internal storage.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.
param-value	string	The input value of the IN or INOUT parameter.

Return Values

Boolean

Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

Scenario #1 — sample code for db-proc-param-assign

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
     (if (db-proc-execute hdbc hstmt)
       ...
       (display (db-get-error-str hdbc))
     )
     (display (db-get-error-str hdbc))
    )
    (display (db-get-error-str hdbc))
   )
   ...
   ...
   (db-logout hdbc)
 )
 (display (db-get-error-str hdbc))
)
```

Scenario #2 — sample code for db-proc-param-assign with
multiple input arguments

```
(if (db-login hdbc dsn uid pwd)
 (begin
  (display "database login succeed !\n")
  (define hstmt (db-proc-bind hdbc "TEST_PROC"))
  (if (statement-handle? hstmt)
    (if (and
        (db-proc-param-assign hdbc hstmt 0 "5")
        (db-proc-param-assign hdbc hstmt 2 "O'REILLY")
        (db-proc-param-assign hdbc hstmt 7 "1998-11-22 12:34:56")
        (db-proc-param-assign hdbc hstmt 8 "1A2B78F0")
       )
       (if (db-proc-execute hdbc hstmt)
        ...
        ...
       )
       (display (db-get-error-str hdbc))
    )
    (display (db-get-error-str hdbc))
  )
  ...
  ...
 )
 (display (db-get-error-str hdbc))
)
```

Notes

- The value for the param-value argument should be entered as a string, without enclosure in single quotation marks (') for SQL_CHAR and SQL_VARCHAR.

- The literal value for SQL_BINARY and SQL_VARBINARY should be a hexadecimal string. Refer to Example #2 on the previous page.

db-proc-param-value

Syntax

```
(db-proc-param-value connection-handle statement-handle param-index)
```

Description

db-proc-param-value retrieves the value of the **OUT** or **INOUT** parameter.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

string
Returns a string which represents the value of the **OUT** or **INOUT** parameter.

Boolean
If unsuccessful, returns **#f** (false)**.** Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
  (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
      (if (db-proc-execute hdbc hstmt)
       (begin
         (define col-count (db-proc-column-count hdbc hstmt))
         (if (and (number? col-count) (> col-count 0))
         (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
   hstmt)))
             ((boolean? result))
             (display result)
             (newline)
          )
         )
         (define prm-count (db-proc-param-count hdbc hstmt))
         (do ((i 0 (+ i 1))) ((= i prm-count))
          (if (not (equal? (db-proc-param-io hdbc hstmt i) "IN"))
            (begin
              (display "output parameter ")
              (display (db-proc-param-name hdbc hstmt i))
              (display " = ")
              (display (db-proc-param-value hdbc hstmt i))
```

```
              (newline)
            )
          )
        )
        ...
        ...
      )
      (display (db-get-error-str hdbc))
    )
    (display (db-get-error-str hdbc))
  )
  (display (db-get-error-str hdbc))
)
...
...
)
(display (db-get-error-str hdbc)
)
```

Notes

The parameter value is unavailable until the user retrieves all the result sets returned from the stored procedure.

db-proc-execute

Syntax

```
(db-proc-execute connection-handle statement-handle)
```

Description

db-proc-execute executes a stored procedure.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
     (if (db-proc-execute hdbc hstmt)
       ...
       (display (db-get-error-str hdbc))
     )
     (display (db-get-error-str hdbc))
   )
    (display (db-get-error-str hdbc))
   )
   ...
   ...
   (db-logout hdbc)
 )
 (display (db-get-error-str hdbc))
 )
```

Notes

The default precision for number or real type is 38 for a column in the table. This is important when executing a stored procedure that retrieves values from that column in the table. The **db-proc-execute** function fails if the exponential part of the value is larger than 38.

For example:

- 1.555E+38 is acceptable
- 1.55E+39 prevents the successful retrieval of the column values

db-proc-fetch

Syntax

```
(db-proc-fetch connection-handle statement-handle)
```

Description

db-proc-fetch retrieves the column values of the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

vector and Boolean
Returns a vector containing all the column values and at the end of the "fetch cycle" returns **#t** (true) when no more records are available to "fetch."

Boolean
If unsuccessful, this function returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

Scenario #1 — sample code for db-proc-fetch

```
(if (db-login hdbc dsn uid pwd)
  (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
     (if (db-proc-param-assign hdbc hstmt 0 "5")
      (if (db-proc-execute hdbc hstmt)
        (begin
         (define col-count (db-proc-column-count))
         (if (and (number? col-count) (> col-count 0))
         (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
hstmt)))
           ((boolean? result) (begin (display result) (newline)))
           (display result
           (newline)
          )
        )
        ...
        ...
       )
            (display (db-get-error-str hdbc)
     )
           (display (db-get-error-str hdbc)
```

```
      )
            (display (db-get-error-str hdbc)
      )
      ...
      ...
    )
    (display (db-get-error-str hdbc)
  )
```

Scenario #2 — Handling multiple result sets for db-proc-fetch

```
  (if (db-login hdbc dsn uid pwd)
  (begin
    (display "database login succeed !\n")
    (define hstmt (db-proc-bind hdbc "MULTI_RESULT"))
    (if (statement-handle? hstmt)
     (if (db-proc-param-assign hdbc hstmt 0 "5")
       (if (db-proc-execute hdbc hstmt)
        (begin
          (do ((col-count (db-proc-column-count hdbc hstmt) (db-proc-
column-count hdbc hstmt)))
             ((or (not (number? col-count)) (= col-count 0)))
           (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
hstmt)))
             ((boolean? result)
             (display result)
             (newline)
            )
           )
          ...
          ...
          )
         (display (db-get-error-str hdbc))
        )
        (display (db-get-error-str hdbc))
      )
      (display (db-get-error-str hdbc))
    )
    ...
    ...
  )
   (display (db-get-error-str hdbc)
  )
```

db-proc-fetch-cancel

Syntax

```
(db-proc-fetch-cancel connection-handle statement-handle)
```

Description

db-proc-fetch-cancel terminates the current "fetch" cycle.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

Scenario #1 — Sample code for db-proc-fetch-cancel

```
(if (db-login hdbc dsn uid pwd)
  (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
     (if (db-proc-param-assign hdbc hstmt 0 "5")
      (if (db-proc-execute hdbc hstmt)
        (begin
         (define col-count (db-proc-column-count))
         (if (and (number? col-count) (> col-count 0))
          (db-proc-fetch-cancel hdbc hstmt)
        )
       )
       ...
       ...
      )
          (display (db-get-error-str hdbc)
    )
      (display (db-get-error-str hdbc))
   )
     (display (db-get-error-str hdbc))
    )
   ...
   ...
   )
   (display (db-get-error-str hdbc))
 )
```

Scenario #2 — Handling multiple result sets for db-proc-fetch-cancel

```
(if (db-login hdbc dsn uid pwd)
(begin
  (display "database login succeed !\n")
  (define hstmt (db-proc-bind hdbc "MULTI_RESULT"))
  (if (statement-handle? hstmt)
   (if (db-proc-param-assign hdbc hstmt 0 "5")
     (if (db-proc-execute hdbc hstmt)
      (begin
        (do ((col-count (db-proc-column-count hdbc hstmt)
         (db-proc-column-count hdbc hstmt)))
          ((or (not (number? col-count)) (= col-count 0)))
          (db-proc-fetch-cancel hdbc hstmt)
        )
        ...
        ...
       )
       (display (db-get-error-str hdbc))
      )
      (display (db-get-error-str hdbc))
    )
    (display (db-get-error-str hdbc))
   )
   ...
   ...
 )
  (display (db-get-error-str hdbc)
)
```

Notes

Multiple result sets can be returned from a single stored procedure. However, each **db-proc-fetch-cancel** call cancels only the current record set. If you want to cancel all result sets, you must call this function for each result set. See example #2 on the previous page.

db-proc-column-count

Syntax

```
(db-proc-column-count connection-handle statement-handle)
```

Description

db-proc-column-count retrieves the number of columns in the return result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

number
Returns a number greater than zero (0) when the record set is available.

Boolean
If no record set is available, the return value is **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
     (if (db-proc-execute hdbc hstmt)
      (begin
       (define col-count (db-proc-column-count))
       (if (and (number? col-count) (> col-count 0))
        (do ((i 0 (+ i 1))) ((= i col-count))
          (display "column ")
          (display (db-proc-column-name hdbc hstmt i))
          (display ": type = ")
          (display (db-proc-column-type hdbc hstmt i))
          (newline)
        )
        (display (db-get-error-str hdbc))
       )
       ...
       ...
      )
      (display (db-get-error-str hdbc))
    )
    (display (db-get-error-str hdbc))
   )
```

```
     (display (db-get-error-str hdbc))
 )
 ...
 ...
 )
 (display (db-get-error-str hdbc)
)
```

Scenario #2 — Handling multiple result sets for db-proc-column-count

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "MULTI_RESULT"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
      (if (db-proc-execute hdbc hstmt)
       (begin
         (do ((col-count (db-proc-column-count hdbc hstmt)
          (db-proc-column-count hdbc hstmt)))
           ((or (not (number? col-count)) (= col-count 0)))
          (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
hstmt)))
            ((boolean? result)
            (display result)
            (newline)
           )
          )
         ...
         ...
         )
         (display (db-get-error-str hdbc))
        )
        (display (db-get-error-str hdbc))
      )
     (display (db-get-error-str hdbc))
    )
    ...
    ...
 )
   (display (db-get-error-str hdbc)
)
```

db-proc-column-name

Syntax

```
(db-proc-column-name connection-handle statement-handle column-index)
```

Description

db-proc-column-name retrieves the name string of the specified column in the result set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
column-index	string	SQL datatype of the specified column in the results set --0 to db-proc-column-count minus 1.

Return Values

string
Returns the name string if successful.

Boolean
If unsuccessful, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
     (if (db-proc-execute hdbc hstmt)
      (begin
       (define col-count (db-proc-column-count))
       (if (and (number? col-count) (> col-count 0))
        (do ((i 0 (+ i 1))) ((= i col-count))
          (display "column ")
          (display (db-proc-column-name hdbc hstmt i))
          (display ": type = ")
          (display (db-proc-column-type hdbc hstmt i))
          (newline)
        )
        (display (db-get-error-str hdbc))
      )
      ...
      ...
    )
```

```
        (display (db-get-error-str hdbc))
      )
      (display (db-get-error-str hdbc))
    )
    (display (db-get-error-str hdbc))
  )
    ...
    ...
  )
   (display (db-get-error-str hdbc)
)
```

db-proc-column-type

Syntax

```
(db-proc-column-type connection-handle statement-handle column-index)
```

Description

db-proc-column-type retrieves the SQL datatype of the specified column in the record set.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
column-index	integer	SQL datatype of the specified column in the record set --0 to db-proc-column-count minus 1.

Return Values

string
Returns a string of SQL datatype when successful.

Boolean
If unsuccessful, returns **#f** (false)**.** Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
   (if (statement-handle? hstmt)
    (if (db-proc-param-assign hdbc hstmt 0 "5")
     (if (db-proc-execute hdbc hstmt)
      (begin
       (define col-count (db-proc-column-count))
       (if (and (number? col-count) (> col-count 0))
        (do ((i 0 (+ i 1))) ((= i col-count))
          (display "column ")
          (display (db-proc-column-name hdbc hstmt i))
          (display ": type = ")
          (display (db-proc-column-type hdbc hstmt i))
          (newline)
        )
        (display (db-get-error-str hdbc))
       )
       ...
       ...
      )
```

```
              (display (db-get-error-str hdbc))
            )
            (display (db-get-error-str hdbc))
          )
          (display (db-get-error-str hdbc))
        )
        ...
        ...
      )
      (display (db-get-error-str hdbc)
    )
```

db-proc-return-exist

Syntax

```
(db-proc-return-exist connection-handle statement-handle)
```

Description

db-proc-return-exist determines whether or not the stored procedure has a return value.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

Boolean

Returns **#t** (true) if a return value exists or **#f** (false) when no return value exists or an error occurs. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
    (if (statement-handle? hstmt)
     (if (db-proc-param-assign hdbc hstmt 0 "5")
      (if (db-proc-execute hdbc hstmt)
       (begin
         (define col-count (db-proc-column-count))
         (if (and (number? col-count) (> col-count 0))
         (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
hstmt)))
             ((boolean? result))
             (display result)
             (newline)
           )
         )
         (if (db-proc-return-exist hdbc hstmt)
          (begin
           (display "return type = ")
           (display (db-proc-return-type hdbc hstmt))
           (newline)
           (display " return value = ")
           (display (db-proc-return-value hdbc hstmt))
           (newline)
          )
         )
         ...
         ...
```

```
            )
            (display (db-get-error-str hdbc))
          )
          (display (db-get-error-str hdbc))
        )
        (display (db-get-error-str hdbc))
      )
    ...
    ...
 )
   (display (db-get-error-str hdbc)
)

)
            (display (db-get-error-str hdbc))
          )
          (display (db-get-error-str hdbc))
        )
        (display (db-get-error-str hdbc))
      )
    ...
    ...
  )
```

Notes

Stored procedures always return an integer value called a return status. This status indicates that the procedure completed successfully or shows the reason for failure. The db-proc-return-exist function always returns **#t** (true).

db-proc-return-type

Syntax

```
(db-proc-return-type connection-handle statement-handle)
```

Description

db-proc-return-type determines the SQL datatype for the return value.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

string
Returns a SQL datatype string, i.e., SQL_VARCHAR.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
      (if (db-proc-param-assign hdbc hstmt 0 "5")
       (if (db-proc-execute hdbc hstmt)
        (begin
          (define col-count (db-proc-column-count))
          (if (and (number? col-count) (> col-count 0))
          (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
hstmt)))
             ((boolean? result))
             (display result)
             (newline)
           )
          )
          (if (db-proc-return-exist hdbc hstmt)
           (begin
            (display "return type = ")
            (display (db-proc-return-type hdbc hstmt))
            (newline)
            (display " return value = ")
            (display (db-proc-return-value hdbc hstmt))
            (newline)
           )
          )
          ...
          ...
         )
        (display (db-get-error-str hdbc))
       )
```

```
        (display (db-get-error-str hdbc))
        )
        (display (db-get-error-str hdbc))
        )
        ...
        ...
    )
    (display (db-get-error-str hdbc)
)
```

Notes

The return value is always an SQL_INTEGER datatype.

db-proc-return-value

Syntax

```
(db-proc-return-value connection-handle statement-handle)
```

Description

db-proc-return-value retrieves the return value (return status) for the stored procedure.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.

Return Values

string
Returns a string which represents the return value.

Throws

None.

Examples

```
(if (db-login hdbc dsn uid pwd)
 (begin
   (display "database login succeed !\n")
   (define hstmt (db-proc-bind hdbc "TEST_PROC"))
     (if (statement-handle? hstmt)
      (if (db-proc-param-assign hdbc hstmt 0 "5")
       (if (db-proc-execute hdbc hstmt)
        (begin
          (define col-count (db-proc-column-count))
          (if (and (number? col-count) (> col-count 0))
          (do ((result (db-proc-fetch hdbc hstmt) (db-proc-fetch hdbc
hstmt)))
             ((boolean? result))
             (display result)
             (newline)
           )
          )
          (if (db-proc-return-exist hdbc hstmt)
           (begin
            (display "return type = ")
            (display (db-proc-return-type hdbc hstmt))
            (newline)
            (display " return value = ")
            (display (db-proc-return-value hdbc hstmt))
            (newline)
           )
          )
          ...
          ...
         )
        (display (db-get-error-str hdbc))
       )
```

```
         (display (db-get-error-str hdbc))
      )
      (display (db-get-error-str hdbc))
   )
 ...
 ...
)
 (display (db-get-error-str hdbc)
)
```

Notes

- Stored procedures can return an integer value called a return status. This status indicates that the procedure completed successfully or shows the reason for failure. SQL Server has a defined set of return values; or users can define their own return values.

- The SQL Server reserves 0 to indicate a successful return, and negative values in the range of -1 to -99 are assigned to a list of reasons for failure. The Numbers 0 and -1 to -14 are in use currently.

Value	Meaning
0	procedure executed without error
-1	missing object
-2	datatype error
-3	process was chosen as deadlock victim
-4	permission error
-5	syntax error
-6	miscellaneous user error
-7	resource error, such as out of space
-8	non-fatal internal problem
-9	system limit was reached
-10	fatal internal inconsistency
-11	fatal internal inconsistency
-12	table or index is corrupt
-13	database is corrupt
-14	hardware error

5.3.6 Message Event Functions

This section contains descriptions of the APIs used to build and manage the DART Event Type Definition. These functions are presented as a group, as they are the functions most used in DART. The current suite of Monk structure functions included within the library **stc_monkfilesys.dll** are:

- **db-struct-bulk-insert** on page 181
- **db-struct-execute** on page 182

db-struct-bulk-insert

Syntax

```
(db-struct-bulk-insert connection-handle table-path)
```

Description

db-struct-bulk-insert inserts an Event Type Definition with repeating nodes (e.g., multiple records) into a table.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	path	A path which represents a table.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Notes

- The Event type MUST be a fixed-length, which can be generated using **dbstruct** with the -f option.

- The number of records that can be inserted into a table is dependent on the memory available and cannot be greater than 32512.

- The format of the literal value of the SQL_DECIMAL and SQL_TIMESTAMP datatype is dependent on the national language support parameter of the SQL server. You can use the SQL statement ALTER SESSION to modify the date format and the decimal character. For example:

  - alter session set NLS_DATE_FORMAT= 'DD-MON-YY'

  - alter session set NLS_NUMERIC_CHARACTERS = '.,'

db-struct-execute

Syntax

```
(db-struct-execute connection-handle statement-handle statement-path)
```

Description

db-struct-execute calls the dynamic statement using the value from the *statement-path* node of the DART Event Type Definition, retrieves all dynamic statement output and places this information into the DART Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
statement-path	statement path	The absolute path to the statement nodes in the Event Type Definition.

Return Values

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

Throws

None.

db-struct-call

Syntax

```
(db-struct-call connection-handle statement-handle procedure-path)
```

Description

db-struct-call calls the stored procedure using the value from the *procedure-path* node of the Event Type Definition, retrieves all procedure output and places this information into the Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
statement-handle	statement handle	The statement handle that identifies the stored procedure specified. This is the handle produced by db-proc-bind.
procedure-path	path	The absolute path to the procedure nodes in the Event Type Definition.

Return Values

Boolean
Returns **#t** (true) if successful; otherwise, returns **#f** (false)**.** Use **db-get-err-str** to retrieve the error message.

db-struct-insert

Syntax

```
(db-struct-insert connection-handle table-path)
```

Description

db-struct-insert composes and executes an SQL INSERT statement according to the information and data carried under the **table-path** node of an Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	event path	A table node of an Event Type Definition.

Return Values

Boolean

Returns **#t** (true) if the execution of the SQL INSERT statement is successful; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
(define input-event-format-file-name "in.ssc")
(define output-event-format-file-name "out.ssc")
(load "in.ssc")
(load "out.ssc")
(define src-collapsed-nodes '())
(define dest-collapsed-nodes '())
(define collapsed-rules '())
(define xlate
  (let ((input ($make-event-map in-delm in-struct))
        (output ($make-event-map out-delm out-struct)))
      (lambda ($make-event-string)
        ($event-parse input $make-event-string)
        ($event-clear output)
        (begin
          (if (db-struct-insert hdbc ~input%in.dbo.table2)
            (begin
              ...
            )
            (begin
              (display (db-get-error-str hdbc))
              (newline)
            )
          )
          ...
          (insert "" ~output%out "")
        )
      )
    )
  )
```

Explanation

The example above shows a typical code segment of a Collaboration Rule that uses the Event Type Definition. In this example, the input defined by **in.ssc** is an Event Type Definition. After parsing the Input Event-string with the Input Event Definition, the Collaboration procedure uses **db-struct-insert** to issue an SQL INSERT statement based on the information carried under Event-path **~input%in.dbo.table2**.

db-struct-update

Syntax

```
(db-struct-update connection-handle table-path where-clause)
```

Description

db-struct-update composes and executes an SQL UPDATE statement according to the information and data carried under the table-path node of an Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	event path	A table node of an Event Type Definition
where-clause	string	The where clause of the SQL SELECT statement.

Return Values

Boolean

Returns **#t** (true) if the execution of the SQL UPDATE statement is successful, or **#f** (false) if the execution of the SQL UPDATE statement fails. Use **db-get-err-str** to retrieve the error message.

Throws

None.

Examples

```
(define input-event-format-file-name "in.ssc")
(define output-event-format-file-name "out.ssc")
(load "in.ssc")
(load "out.ssc")
(define src-collapsed-nodes '())
(define dest-collapsed-nodes '())
(define collapsed-rules '())
(define xlate
  (let ((input ($make-event-map in-delm in-struct))
        (output ($make-event-map out-delm out-struct)))
    (lambda ($make-event-string)
      ($event-parse input $make-event-string)
      ($event-clear output)
      (begin
        (if (db-struct-update hdbc ~input%in.dbo.table2 "ID = 5")
          (begin
            ...
          )
          (begin
            (display (db-get-error-str hdbc))
            (newline)
          )
        )
        ...
        (insert "" ~output%out "")
      )
```

```
            )
        )
    )
```

Explanation

The example above shows a typical code segment of a Collaboration Rule that uses the Event Type Definition. In this example, the input defined by **in.ssc** is an Event Type Definition. After parsing the input Event-string with the Input Event Type Definition, the Collaboration procedure uses **db-struct-update** to issue an SQL UPDATE statement based on the information carried under the Event-path **~input%in.dbo.table2.**

db-struct-select

Syntax

```
(db-struct-select connection-handle table-path where-clause)
```

Description

db-struct-select composes and executes an SQL SELECT statement according to the information and data carried under the table-path node of a DART Event Type Definition.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	event path	A table node of an Event Type Definition
where-clause	string	The where clause of the SQL SELECT statement.

Return Values

Boolean
Returns **#t** (true) if the execution of the SQL SELECT statement is successful; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Throws

None.

Remarks

- Both **db-struct-select**, and **db-struct-fetch** use the same algorithm to generate the selection name for the **db-sql-select** and **db-sql-fetch** procedure call. If the table path is a table node under an owner (schema) node the selection name will be **owner.table**.

- If the table path does not have an owner node above it, the selection name will be **table**. You must issue a **db-sql-fetch-cancel** call with either **owner.table** or **table** as the selection name, if you want to cancel the selection.

Examples

```
(define input-event-format-file-name "in.ssc")
(define output-event-format-file-name "out.ssc")
(load "in.ssc")
(load "out.ssc")
(define src-collapsed-nodes '())
(define dest-collapsed-nodes '())
(define collapsed-rules '())
(define xlate
  (let ((input ($make-event-map in-delm in-struct))
        (output ($make-event-map out-delm out-struct)))
    (lambda ($make-event-string)
      ($event-parse input $make-event-string)
      ($event-clear output)
      ($event-parse output (event->string output))
```

```
(begin
  (if (db-struct-select hdbc ~output%out.dbo.table2 "ID = 5")
    (begin
      (db-struct-fetch hdbc ~output%out.dbo.table2)
      ...
       (db-sql-fetch-cancel hdbc "dbo.table2")
    )
    (begin
      (display (db-get-error-str hdbc))
      (newline)
    )
  )
  ...
  (insert "" ~output%out "")
)
)
)
)
```

Explanation

The example above shows a typical code segment of a DART Collaboration Rules file that uses the DART Event Type Definition. In this example, the output defined by **out.ssc** is a DART Event Type Definition. After clearing the Output Event-string, the Collaboration Service uses **db-struct-select** to issue an SQL SELECT statement based on the information carried under the Event-path **~output%out.dbo.table2**. The selection was cancelled by **db-sql-fetch-cancel** with **dbo.table2** as the selection name.

db-struct-fetch

Syntax

```
(db-struct-fetch connection-handle table-path)
```

Description

db-struct-fetch composes and executes an SQL FETCH statement according to the information and data carried under the table-path node of a DART Event Type Definition, and stores the return column values inside each of the column nodes.

Parameters

Name	Type	Description
connection-handle	connection handle	A connection handle to the database.
table-path	event path	A table node of an Event Type Definition

Return Values

Path
Returns the table path if the execution of the SQL FETCH statement is successful, or

Boolean
Returns **#t** (true) when the end of the fetch cycle is reached; otherwise, returns **#f** (false). Use **db-get-err-str** to retrieve the error message.

Examples

```
(define input-event-format-file-name "in.ssc")
(define output-event-format-file-name "out.ssc")
(load "in.ssc")
(load "out.ssc")
(define src-collapsed-nodes '())
(define dest-collapsed-nodes '())
(define collapsed-rules '())
(define xlate
  (let ((input ($make-event-map in-delm in-struct))
        (output ($make-event-map out-delm out-struct)))
    (lambda ($make-event-string)
      ($event-parse input $make-event-string)
      ($event-clear output)
      (begin
        (if (db-struct-select hdbc ~output%out.dbo.table2)
            (do ((result "") ((boolean? result))
              (set! result (db-struct-fetch hdbc
~output%out.dbo.table2))
              (if (boolean? result))
                (if (not result)
                  (begin
                    (display "db-struct-fetch failed!\n")
                    (display (db-get-error-str hdbc))
                    (newline)
                  )
                  (begin
                    ...
                  )
                )
```

```
                (begin
                  (display result)
                  (newline)
                )
              )
            )
          )
          ...
          (insert "" ~output%out "")
        )
      )
    )
  )
```

Explanation

The example above shows a typical code segment of a Collaboration Rule that uses the Event Type Definition. In this example, the output defined by **out.ssc** is an Event Type Definition. After clearing the Output Event-string with the Output Event Type Definition, the Collaboration procedure uses **db-struct-select** to issue an SQL SELECT statement based on the information carried under Event- path **~output%out.dbo.table2**.

It repeatedly uses **db-struct-fetch** to issue the SQL FETCH statement and store the resulting column values inside each column node under the table path **~output%out.dbo.table2** until there are no more records to fetch.

Index