SeeBeyond ICAN Suite

e*Way Intelligent Adapter for JDBC/ODBC User's Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Version 20050406093843.

Contents

Chapter 5

JDBC/ODBC e*Way Methods 143

Chapter 1

Introduction

This document describes how to install and configure the e*Way Intelligent Adapter for JDBC/ODBC and ODBC.

This introduction includes the following:

1.1 JDBC/ODBC e*Way Overview

The JDBC/ODBC e*Way enables the e*Gate system to exchange data with external databases. This document describes how to install and configure the JDBC/ODBC e*Way.

This section contains the following information related to the JDBC/ODBC e*Way:

- **Driver types**
- **Components**
- **Operational overview**
- **Supported operating systems**
- **System requirements**

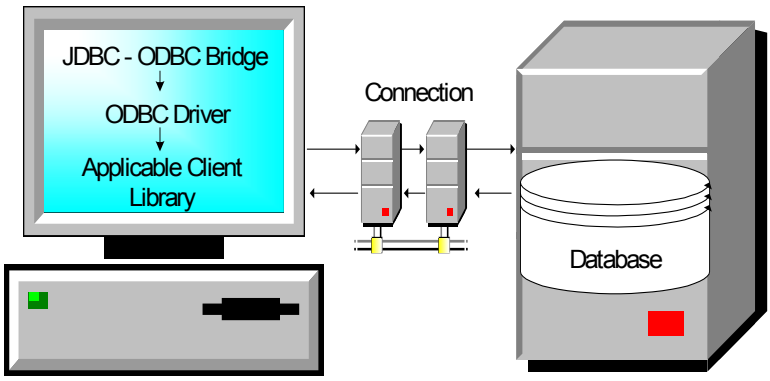
1.1.1 Driver Types

There are four different types of JDBC technology drivers. They are described in the following sections.

Type One Driver

A JDBC/ODBC bridge provides JDBC API access through one or more ODBC drivers. Some ODBC native code and in many cases native database client code must be loaded on each client machine that uses this type of driver.

Figure 1 Typical Type 1 Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available

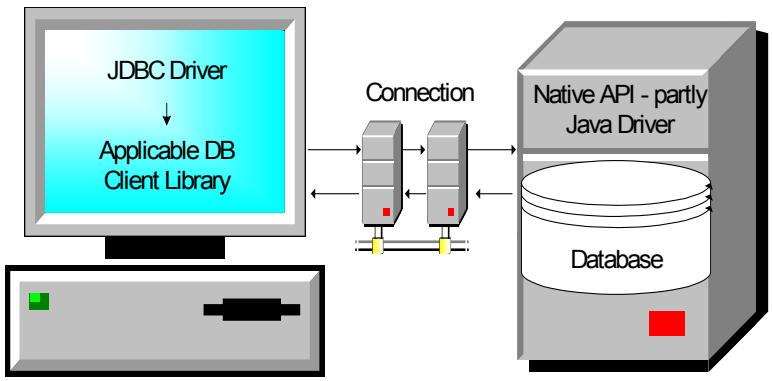
Cons

- Performance is degraded since the JDBC call goes through the bridge to the ODBC driver then to the native database connectivity interface. The results are then sent back through the reverse process
- Limited Java feature set
- May not be suitable for a large-scale application

Type Two Driver

A native-API partly Java technology-enabled driver converts JDBC calls into calls on the client API for DBMSs. Like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

Figure 2 Typical Type 2 Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge
- Limited Java feature set

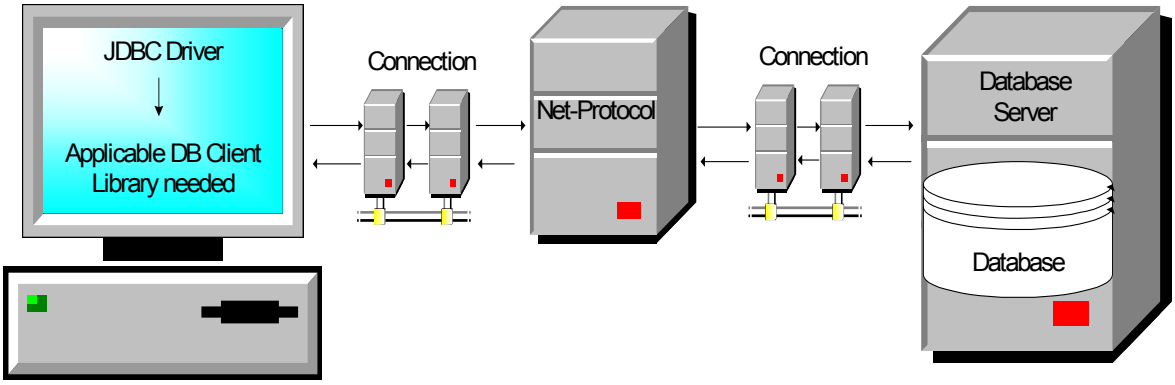
Cons

- Applicable Client library must be installed
- Type 2 driver shows lower performance than type 3 or 4

Type Three Driver

A net-protocol fully Java-enabled driver translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect all of its Java technology-based clients to many different databases.

Figure 3 Typical Type 3 Middleware Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge and Type 2 Drivers
- Advanced Java feature set
- Scalable
- Caching
- Advanced system administration

- Does not require applicable database client libraries

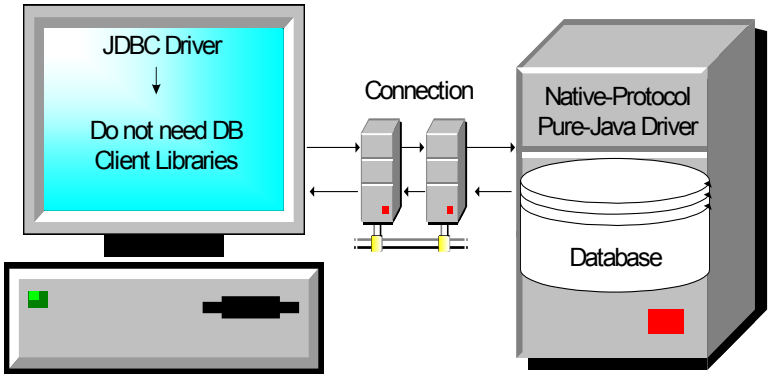
Cons

- Requires a separate JDBC middleware server to translate specific native-connectivity interface.

Type Four Driver

A native-protocol fully Java technology-enabled driver converts JDBC technology calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server.

Figure 4 Typical Type 4 Driver Configuration



The pros and cons for using this type of driver are as follows:

Pros

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge and Type 2 Drivers
- Scalable
- Caching
- Advanced system administration
- Superior performance
- Advance Java feature set
- Does not require applicable database client libraries

Cons

- Each database will require a driver

1.1.2 Components

The following components comprise the JDBC/ODBC e*Way:

- **e*Way Connections:** The database e*Way Connections provide access to the information necessary for connecting to a specified external system.
- **stcjdbcx.jar**: Contains the logic required by the e*Way to interact with the external databases.

A complete list of installed files appears in **Table 1 on page 23**.

1.1.3 Operational Overview

The JDBC/ODBC e*Way uses Java Collaborations to interact with one or more external databases. By using the Java Collaboration Service it is possible for e*Gate components—such as e*Way Intelligent Adapters (e*Ways) and Business Object Brokers (BOBs)—to connect to external databases and execute business rules written entirely in Java.

An e*Gate component is defined as *Java-enabled* based on the selection of the Java Collaboration Service in the Collaboration Rule setup. For more information on the Java Collaboration Service, see **"The Java Collaboration Service" on page 68**.

1.2 ODBC e*Way Overview

SeeBeyond™ developed the e*Way Intelligent Adapter for ODBC as a graphically-configurable e*Way. The ODBC 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 ODBC 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 contains instructions for installing and configuring the ODBC e*Way. This overview section includes information about the following as they relate to the ODBC e*Way:

- **Using SQL**
- **Components**
- **Supported operating systems**
- **System requirements**
- **External system requirements**

1.2.1 Components

The ODBC e*Way is comprised of the following:

- **stcewgenericmonk.exe**, the executable component
- Configuration files, which the e*Way Editor uses to define configuration parameters
- Monk external function scripts
- e*Way Monk functions

A complete list of installed files appears in **Table 2 on page 26** and **Table 3 on page 27**.

1.3 System Requirements

The performance and functionality of the JDBC/ODBC e*Way depends on the driver(s) selected. Certain drivers may not support all JDBC features. Consult the documentation for your respective driver(s) for more information.

To use the JDBC/ODBC e*Way, you need the following:

- An e*Gate Participating Host.
- A TCP/IP network connection.

Host System Requirements

The external system requirements are different for a GUI host machine—specifically a machine running the ETD Editor and the Java Collaboration Editor GUIs—versus a participating host which is used solely to run the e*Gate schema.

GUI Host Requirements

To enable the GUI editors to communicate with the external system, the following items must be installed on any host machines running the GUI editors:

- If you are using driver types 1, 2, or 3, the client library for your specific database installed on Windows systems utilize the ETD builder.
- ODBC driver.

If the GUI host machine also executes the JDBC/ODBC e*Way, the host machine must also meet the Participating Host requirements.

Participating Host Requirements

The appropriate driver type for your database.

A list of drivers from third party vendors is available at:

http://industry.java.sun.com/products/jdbc/drivers

Note: *Open and review the Readme.txt for any additional requirements prior to installation. The Readme.txt is located on the Installation CD_ROM .*

1.4 Supported Operating Systems

The JDBC/ODBC e*Way is available on the following operating systems:

- Windows 2000, Windows XP, and Windows Server 2003
- HP Tru64 V5.1A
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5.1L and 5.2
- Sun Solaris 8 and 9
- Japanese Windows 2000, Windows XP, and Windows Server 2003
- Japanese HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Japanese IBM AIX 5.1L and 5.2
- Japanese Sun Solaris 8 and 9
- Korean Windows 2000, Windows XP, and Windows Server 2003
- Korean HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- Korean IBM AIX 5.1L and 5.2
- Korean Sun Solaris 8 and 9

Chapter 2

Installation

This chapter describes how to install the JDBC/ODBC and ODBC e*Ways. It includes the following information:

2.1 Installing the JDBC/ODBC e*Way on Windows

2.1.1 Pre-installation

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

2.1.2 Installation Procedure

To install the JDBC/ODBC e*Way on a Windows operating system

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Insert the e*Gate 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 Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

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

Be sure to install the e*Way files in the suggested "client" installation directory. The installation utility detects and suggests the appropriate installation directory. **Unless**

you are directed to do so by STC support personnel, do not change the suggested "installation directory" setting.

2.2 Installing the JDBC/ODBC e*Way on UNIX

2.2.1 Pre-installation

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

2.2.2 Installation Procedure

To install the JDBC/ODBC 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.

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 Files/Directories Created by the Installation

The JDBC/ODBC e*Way installation process installs the following files within the e*Gate directory tree. Files are installed within the "egate\client" tree on the Participating Host and committed to the "default" schema on the Registry Host.

Table 1 Files created by the installation—JDBC/ODBC

e*Gate Directory	File(s)
classes	stcjdbcx.jar
configs\jdbcodbc\	jdbcodbc.def

Table 1 Files created by the installation—JDBC/ODBC

e*Gate Directory	File(s)
etd\	stcewjdbcx.ctl jdbcodbc.ctl dbwizard.ctl db.ctl
etd\db\	Com_stc_jdbcx_dbcfg.java Com_stc_jdbcx_dbcfg.xsc emptyETD.xsc emptyETD.xbs emptyETD.jar
ThirdParty\merant\classes	DGbase.jar spy.jar
ThirdParty\sun\classes	jdbc2_0-stdext.jar

2.4 Monk ODBC Installation

2.4.1 Installation Overview

The installation procedure depends upon the operating system of the Participating Host on which you are installing this e*Way. You must have Administrator privileges to install this e*Way on Windows systems or UNIX.

2.4.2 Installation Decisions

This section presents decisions to be made before beginning the installation. These decisions apply to UNIX or Windows systems:

1. The operating system/platform on which the ODBC e*Way will operate.
2. The database network software required to operate the ODBC e*Way.

   For Oracle:

   - SQL *Net8

   For Sybase:

   - Open Client version11.1.x or 12
3. The Oracle networking options to be installed.

   On UNIX:

   - SQL *Net8
   - TCP/IP Protocol Adaptor

   On Windows:

   - SQL *Net8

- TCP/IP Adapter
- OCI

Issue the following command to determine which version of SQL *Net is installed:

On UNIX:

```
echo $ORACLE_HOME
/opt/oracle/app/oracle/product/8.1.6
```

The output shows that SQL *Plus Version 8.1.6 is installed.

2.4.3 Installing Client and Network Components on Windows

The following Networking Options must be installed and configured before running the ODBC e*Way:

- The Oracle8 or Oracle8i Oracle Client

Note: *The Oracle Client is not required for the ODBC e*Way to communicate with Microsoft SQL Server. The Oracle Client is required in order to communicate with any other database. If you do use an Oracle Client, the 32-bit version is required. The 64-bit Oracle Client is not compatible with this e*Way.*

- SQL*Net8 for Oracle8 and 8i
- TCP/IP Protocol Adapter
- OCI (Oracle Call Interface)
- The Sybase Open Client

2.5 Installing the ODBC e*Way on Windows

2.5.1 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.

2.5.2 Installation Procedure

To install the ODBC e*Way on a Windows system

1. Log in as an Administrator on the workstation on which you want to install the e*Way.
2. Insert the e*Way installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's "Autorun" feature is enabled, the setup application should launch automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or

the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.

4 The InstallShield setup application launches. 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.***

The ODBC 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.

Table 2 Installation Directories and Files (Windows)—ODBC

Install Directory	Files
bin\	stcewgenericmonk.exe stcstruct.exe stc_dbapps.dll stc_dbmonkext.dll stc_dbodbc.dll stccdbctest.exe
configs\stcewgenericmonk\	dart.def dartRule.txt
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 odbcmsg.ssc odbcmsg-display.monk db_bind.monk db-sanitize-symbol.monk

2.6 Installing the ODBC e*Way on UNIX

2.6.1 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.

2.6.2 Installation Procedure

To install the ODBC 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 **e*Gate Add-on Application** 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. Refer to the installation instructions for e*Gate for the most up-to-date information.

Table 3 Installation Directories and Files (UNIX)—ODBC

Install Directory	Files
bin/	stcewgenericmonk.exe stc_dbapps.dll stc_dbmonkext.dll stc_dbodbc.dll stc_dbctest.exe stcstruct.exe stc_dbodbc.dll
configs/stcewgenericmonk/	dart.def dartRule.txt
monk_library	dart.gui

Table 3 Installation Directories and Files (UNIX)—ODBC

Install Directory	Files
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 odbcmsg.ssc odbcmsg-display.monk db-sanitize-symbol.monk

2.7 DataDirect 4.1 ODBC Drivers

This section covers installation, setup, and testing of the DataDirect ODBC Drivers.

2.7.1 Setting up the Shared Library Search Path

You must set up the shared library search path used by both the ODBC e*Way and the Sybase Open Client. The library search path environment variables are required to be set so that the ODBC core components and drivers can be located at the time of execution.

You can define these environment variables in *.cshrc* in the C shell or *.profile* in the Korn/Bash shell. Follow these scripts when setting up the shared library search path:

Korn/Bash Shell

```
if [ "$LD_LIBRARY_PATH" = "" ]; then
  LD_LIBRARY_PATH=/opt/odbc/lib
else
  LD_LIBRARY_PATH=/opt/odbc/lib:$LD_LIBRARY_PATH
fi
export LD_LIBRARY_PATH
```

C Shell

```
if ($?LD_LIBRARY_PATH == 1) then
  setenv LD_LIBRARY_PATH  /opt/odbc/lib:${LD_LIBRARY_PATH}
else
  setenv LD_LIBRARY_PATH  /opt/odbc/lib
endif
```

2.7.2 Setting up the ODBC Data Source Definition File

In the UNIX environment, there is no ODBC Administrator. To configure a data source, you must edit the odbc.ini file, a plain text file that is normally located in the user's

$HOME directory. This file is maintained using any text editor to define data source entries as described in the "Connecting to a Data Source Using a Connection String" section of each driver's chapter. A sample file (odbc.ini) is located in the driver installation directory.

UNIX support of the database drivers also allows the use of a centralized **.odbc.ini** file that a system administrator can control. This is accomplished by setting the environment variable ODBCINI to point to the fully qualified pathname of the centralized file.

The search order for the location of the **.odbc.ini** file is as follows:

1. Check $ODBCINI
2. Check $HOME/.odbc.ini

There must be an ODBC section in the **.odbc.ini** file that includes the InstallDir keyword. The value of this keyword must be the path to the directory under which the /lib and /messages directories are contained. For example, if you choose the default install directory, then the following line must be in the [ODBC] section:

```
InstallDir=/opt/odbc
```

Sample .odbc.ini File

The following is an **.odbc.ini** file which contains some sample values to use when setting these environment variables.

```
[ODBC Data Sources]
DB2 Wire Protocol=DataDirect 4.00 DB2 Wire Protocol Driver
dBase=DataDirect 4.0 dBaseFile(*.dbf)
Informix=DataDirect 4.0 Informix
Informix Wire Protocol=DataDirect 4.0 Informix Wire Protocol
Oracle=DataDirect 4.0 Oracle
Oracle Wire Protocol=DataDirect 4.0 Oracle Wire Protocol
SQLServer Wire Protocol=DataDirect 4.0 SQL Server Wire Protocol
Sybase Wire Protocol=DataDirect 4.0 Sybase Wire Protocol
Text=DataDirect 4.0 TextFile(*.*)

[DB2 Wire Protocol]
Driver=/opt/odbc/lib/DGdb217.so
Description=DB2 Wire Protocol Driver
LogonID=uid
Password=pwd
DB2AppCodePage=1252
ServerCharSet=1252
IpAddress=db2host
Database=db
TcpPort=50000
Package=db2package
Action=REPLACE
QueryBlockSize=8
CharSubTypeType=SYSTEM_DEFAULT
ConversationType=SINGLE_BYTE
CloseConversation=DEALLOC
UserBufferSize=32
MaximumClients=35
GrantExecute=1
GrantAuthid=PUBLIC
OEMANSI=1
DecimalDelimiter=PERIOD
```

```
DecimalPrecision=15
StringDelimiter=SINGLE_QUOTE
IsolationLevel=CURSOR_STABILITY
ResourceRelease=DEALLOCATION
DynamicSections=32
Trace=0
WithHold=0

[dBase]
Driver=/opt/odbc/lib/DGdbf17.so
Description=dBaseFile(*.dbf)
Database=/opt/odbc/demo
CacheSize=4
Locking=RECORD
CreateType=dBASE5
IntlSort=0
UseLongNames=1
UseLongQualifiers=1
ApplicationUsingThreads=1

[Informix]
Driver=/opt/odbc/lib/DGinf17.so
Description=Informix
Database=db
LogonID=uid
Password=pwd
ServerName=informixserver
HostName=informixhost
Service=online
Protocol=onsoctcp
EnableInsertCursors=0
GetDBListFromInformix=0
CursorBehavior=0
CancelDetectInterval=0
TrimBlankFromIndexName=1
ApplicationUsingThreads=1

[Informix Wire Protocol]
Driver=/opt/odbc/lib/DGifcl17.so
Description=Informix Wire Protocol
Database=db
LogonID=uid
Password=pwd
HostName=informixhost
PortNumber=1500
ServerName=informixserver
EnableInsertCursors=0
GetDBListFromInformix=0
CursorBehavior=0
CancelDetectInterval=0
TrimBlankFromIndexName=1
ApplicationUsingThreads=1

[Oracle]
Driver=/opt/odbc/lib/DGor817.so
Description=Oracle
LogonID=uid
Password=pwd
ServerName=oraclehost
CatalogOptions=0
ProcedureRetResults=0
EnableDescribeParam=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
```

```
[Oracle Wire Protocol]
Driver=/opt/odbc/lib/DGora17.so
Description=Oracle Wire Protocol
LogonID=uid
Password=pwd
HostName=oracleserver
PortNumber=1521
SID=oraclesid
CatalogOptions=0
ProcedureRetResults=0
EnableDescribeParam=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1

[SQLServer Wire Protocol]
Driver=/opt/odbc/lib/DGmsss17.so
Description=SQL Server Wire Protocol
Database=db
LogonID=uid
Password=pwd
Address=sqlserverhost,1433
QuotedId=No
AnsiNPW=No

[Sybase Wire Protocol]
Driver=/opt/odbc/lib/DGase17.so
Description=Sybase Wire Protocol
Database=db
LogonID=uid
Password=pwd
NetworkAddress=serverhost,4100
EnableDescribeParam=1
EnableQuotedIdentifiers=0
OptimizePrepare=1
RaiseErrorPositionBehavior=0
SelectMethod=0
ApplicationUsingThreads=1

[Text]
Driver=/opt/odbc/lib/DGtxt17.so
Description=TextFile(*.*)
Database=/opt/odbc/demo
AllowUpdateAndDelete=0
CacheSize=4
CenturyBoundary=20
FileOpenCache=0
UndefinedTable=GUESS
IntlSort=0
ScanRows=25
TableType=Comma
UseLongQualifiers=0
ApplicationUsingThreads=1

[ODBC]
Trace=0
TraceFile=odbctrace.out
TraceDll=/opt/odbc/odbctrac.so
InstallDir=/opt/odbc
ConversionTableLocation=/opt/odbc/tables
UseCursorLib=0
```

Caution: *The "Trace" value must be set to* ***0****. Setting this value to* ***1*** *can cause some third-party applications to interfere with e*Gate.*

Optional Environment Variables

Many of the drivers must have environment variables set as required by the database client components used by the drivers. Consult the system requirements in each of the individual driver sections for additional information pertaining to individual driver requirements.

2.7.3 The ivtestlib Tool

The ivtestlib tool is provided to help diagnose configuration problems (such as environment variables not correctly set or missing database management system client components) in the UNIX environment. This command attempts to load a specified ODBC driver and prints out all available error information if the load fails. For example, the following command attempts to load the Oracle driver on Solaris.

```
ivtestlib /opt/odbc/lib/dgor816.so
```

The executable [ivtestlib] is located in the `/opt/odbc/bin` directory

2.7.4 Testing the ODBC Driver

To test if the driver is running correctly, log in as the client (e.g., ODBC) and run the test program **stcodbctest.exe**.

```
stcodbctest data_source user_name password
```

Example stcodbctest ODBC e*Way

A typical output message for **stcodbctest** is shown below.

```
Environment handle allocated.
Connection handle allocated.
Data source: SQL4.0 found.
Database connection established.

ODBC Driver Information:
        Driver Name                     : DGSS617.DLL
        Driver Version                  : 04.00.0004
        Driver Manager ODBC Version     : 03.52.0000
        Driver ODBC Version             : 03.51
        Driver ODBC API Conformance     : Level 1 supported
        Driver ODBC SQL Conformance     : Core grammar supported
        Driver ODBC Procedure Support   : Yes

DBMS Product Information:
        DBMS Name                       : Microsoft SQL Server
        DBMS Version                    : 08.00.0194

Data Source Information:
        Data Source Name                : SQL4.0
        Server Name                     : anu2000
        Database Name                   : pubs
        User Name                       : dbo
        Transaction Support             : Both DML and DDL statements
are supported

ODBC Function Information:
        SQLNumResultCols                : supported
        SQLDescribeCol                  : supported
```

```
        SQLBindCol                      : supported
        SQLNumParams                    : supported
        SQLDescribeParam                : not supported
        SQLBindParameter                : supported
        SQLProcedures                   : supported
        SQLProcedureColumns             : supported

Database connection terminated.
Connection handle freed.
```

It is important that all of the above ODBC Function Information parameters indicate that they are supported.

Note: *In order to assure that the latest statement functionality is available, the SQLDescribeParam line item must be present, and indicate "supported".*

2.8 Installing the ODBC Drivers for Compaq

To operate the ODBC e*Way on Compaq Tru64 systems, obtain the Compaq ODBC drivers from the following location:

http://tru64unix.compaq.com/data-access/download.htm

You are required to register prior to downloading the drivers.

2.9 Oracle Network Components

Install the following Oracle networking options when running as a client database.

SQL *Net8 (Oracle8)

TCP/IP Protocol Adapter

Note: *Install SQL *PLUS to test out the connection.*

2.9.1 SQL *Net V2 Configuration Files

Before you can configure SQL *Net8 you must have the following files ready:

- listener.ora
- tnsnames.ora
- sqlnet.ora

Sample Listener configuration file—listener.ora

```
# LISTENER.ORA Configuration
File:/opt/oracle/app/oracle/product/8.1.6/network/admin/listener.ora
# Generated by Oracle configuration tools.

LISTENER =
```

```
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = circe)(PORT = 1521))
      )
    )
    (DESCRIPTION =
      (PROTOCOL_STACK =
        (PRESENTATION = GIOP)
        (SESSION = RAW)
      )
      (ADDRESS = (PROTOCOL = TCP)(HOST = circe)(PORT = 2481))
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /opt/oracle/app/oracle/product/8.1.6)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (ORACLE_HOME = /opt/oracle/app/oracle/product/8.1.6)
      (SID_NAME = orcl816)
    )
  )
```

LISTENER is the default listener name, which is recommended by Oracle in a standard installation that requires only one listener on a machine.

Listener address section ADDRESS specifies what address to listen to. The listener listens for inter-process calls (IPC's) as well as calls from other nodes.

Two IPC addresses are created for each database that a listener listens to. In one, the key value is equal to the service name (for example, finance.world). It is used for connections from other applications on the same node. The other IPC address (for example, orcl) is used by the database dispatcher to identify the listener.

For communication with other nodes, listener listens to the host (for example, finance.company.com) at a particular port (for example, 1521) using the specified protocol (for example, TCP/IP).

The section SID_LIST_LISTENER is used to describe the SID (system identified) of the databases (for example, orcl) on which the listener listens. The service name (for example, finance.world) is used as the global name.

The control parameter STARTUP_WAIT_TIME_LISTENER sets the number of seconds that the listener sleeps before responding to the first listener control status command. This feature assures that a listener with a slow protocol has time to start up before responding to a status request. The default is 0.

CONNECT_TIMEOUT_LISTENER sets the number of seconds that the listener waits to get a valid SQL*Net connection request before dropping the connection.

TRACE_LEVEL_LISTENER indicates the level of detail the trace facility records for listener events. ADMIN is the highest.

Sample Client file—tnsnames.ora

```
# TNSNAMES.ORA Configuration
File:/opt/oracle/app/oracle/product/8.1.6/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

CIRCE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = circe)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = orcl816))
  )
ENIGMA =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = enigma)(PORT = 1521))
    (CONNECT_DATA = (SID = orcl8))
  )
LAMBDA =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(Host = lambda)(Port = 1521))
    (CONNECT_DATA = (SERVICE_NAME = LAMBDA))
  )
```

All connect distributors are assigned service names (for example, finance.world). The user specifies the service name to identify the service to which the user wants to connect. The ADDRESS section specifies the listener address. See **"Sample Listener configuration file—listener.ora" on page 33** for listener address.

The CONNECT_DATA section specifies the SID (system identified) by the remote database.

Network component file—sqlnet.ora

```
File: sqlnet.ora
################
# Filename......: sqlnet.ora
# Node..........: local.world
# Date..........: 24-MAR-98 13:21:20
################
AUTOMATIC_IPC = OFF
TRACE_LEVEL_CLIENT = OFF
names.directory_path = (TNSNAMES)
names.default_domain = world
name.default_zone = world
sqlnet.expire_time = 10
```

The sqlnet.expire_time parameter determines how often SQL*Net sends a probe to verify that a client-server connection is still active. A value of 10 (minutes) is recommended by Oracle.

After you have generated the required configuration files, do the following:

- On the server side, move all three files to:

  `$ORACLE_HOME/network/admin`

- On the client side, distribute tnsnames.ora and sqlnet.ora and put them in:

  `$ORACLE_HOME/network/admin`

Verify that the file/etc/services has the entry LISTENER 1521/tcp.

2.9.2 Testing the SQL *Net Configuration

Before you can use SQL*Net with the server, you need to start a listener on the server. A listener is used by SQL*Net on the server side to receive an incoming connection from SQL*Net clients.

To start a listener, enter the following command in the server:

```
lsnrctl start listener name
```

Where <listener name> is optional for the default listener. For example, to start up the default LISTENER in the machine enterprise, the command would be:

```
lsnrct start
```

When you are running as a client, if the listener starts up successfully, you can use SQL*Plus on the client side to test whether SQL*Net is configured properly by establishing a connection with the server. For example:

```
sqlplus hcaufield/phoebie@oracle.world
```

This command starts up sqlplus in the client machine enterprise and connect to the server specified by oracle.world as user *hcaufield* with password *phoebie*. The syntax of the command is:

```
sqlplus user name/password@service name
```

Note: *The $ORACLE_HOME/network/admin/tnsnames.ora defines the service name for each Oracle data source.*

2.9.3 Troubleshooting Checklist

- Ensure you have protocol-level connectivity (for TCP/IP, connectivity can be tested using the ping utility).
- Ensure client machine has configuration files (TNSNAMES.ORA and SQLNET.ORA) in the $ORACLE_HOME/network/admin directory. Also check that the server has the configuration files (LISTENER.ORA, TNSNAMES.ORA, and SQLNET.ORA) in its default directory.
- Check whether the listener is "listening" for the same protocol the client is trying to connect through.
- Verify that both server and client are running either Net8 or SQL *Net V2. Net 8 and SQL *Net V2 can communicate to each other. Verify the version by running the Oracle Universal Installer.
- Ensure that you have the necessary Net8 protocol support installed. Verify by running the Oracle Universal Installer.
- Verify that the net service name is typed correctly. The net service name should be listed under the Net Service Names folder in the Net8 Assistant.
- check to see if the default domain in your profile is set. If it is, then the net service names will have the same value appended to them. For example, if the default domain in your profile is set to ACME.COM, then all net service names have the .ACME.COM extension appended.

- If you are using TCP/IP, try replacing the HOST name in the net service name address with the IP address of the server machine.

For more information on specific error messages or technical bulletins on errors received when performing these diagnostics tests, refer:

- The Net8 Administrator's Guide

Chapter 3

e*Way Connection Configuration

This chapter describes how to configure the JDBC/ODBC and ODBC e*Way Connections.

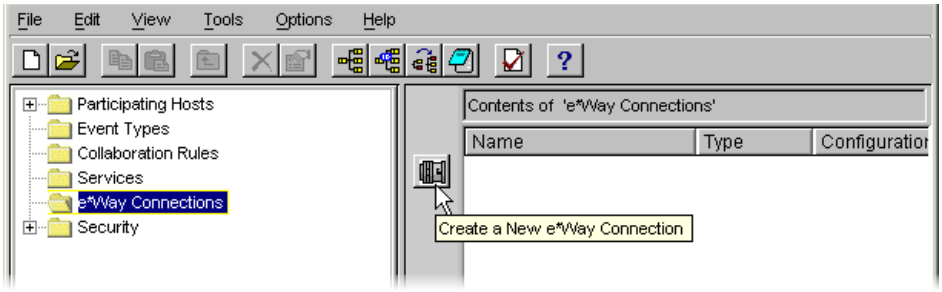
3.1 Create e*Way Connections

The e*Way Connections are created and configured in the Schema Designer.

To create and configure the e*Way Connections

1 In the Schema Designer's Component editor, select the e*Way Connections folder.

Figure 5 The e*Way Connections Folder

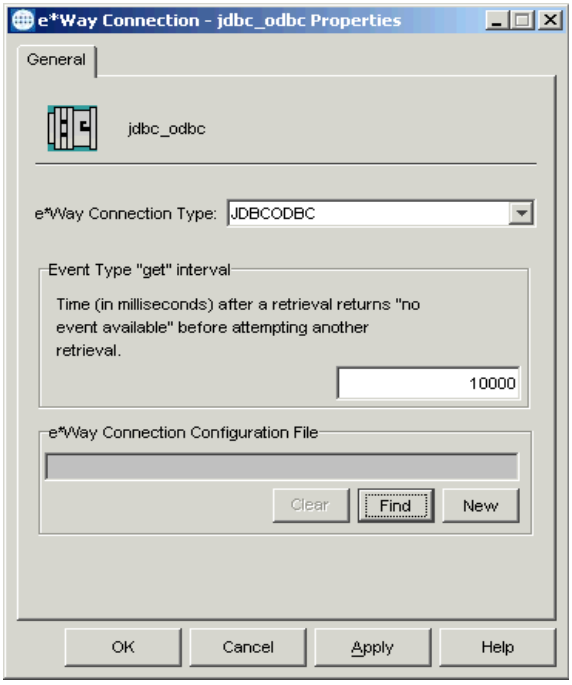


2 On the Palette, click the **New e*Way Connection** icon.

3 The **New e*Way Connection Component** dialog box opens. Enter a name for the e*Way Connection and click **OK**.

4 Double-click the new e*Way Connection to open the e*Way Connection Properties dialog box. See **Figure 6 on page 39**.

Figure 6 e*Way Connection Properties Dialog Box



5 From the e*Way Connection Type drop-down box, select **JDBCODBC**.

6 Enter the **Event Type "get" interval** in the dialog box provided (optional).

7 Click **New** to create a new e*Way Connection Configuration File.

The e*Way Connection Configuration File Editor appears.

The e*Way Connection configuration file parameters are organized into the following sections:

- DataSource Settings
- Connector Settings

3.1.1 DataSource Settings

The DataSource settings define the parameters used to interact with the external database.

class

Description

Specifies the name of the Java class in the JDBC driver that is used to implement the ConnectionPoolDataSource interface.

Required Values

The default is **sun.jdbc.odbc.JdbcOdbcDriver**.

Additional Drivers

If you wish to use any other type of driver, you must add the driver through the Collaboration Editor. To do this, complete the following steps:

1 Copy your choice of drivers to:

```
\ThirdParty\<driver>\classes
```

Where <driver> is the name of your driver provider. For example:

```
\ThirdParty\Merant\classes
```

or

```
\ThirParty\Sun\classes
```

2 To include the above copied files in the classpath, select **Options** from the Collaboration Editor **Tools** menu, select **Options**.

3 In the **Java ClassPaths** window, select the driver files from the list of available files.

4 From the **File** menu in the Collaboration Editor, select **Compile**.

This will enable the .ctl file to find your chosen drivers during run time.

For Users of the Merant Type 4 drivers

If you are using Merant type 4 drivers, do the following:

1 Copy the util.jar, base.jar and <database>.jar files to:

```
\ThirdParty\Merant\classes
```

Where <database>.jar is the name of the your particular database's .jar file.

2 To include the above copied files in the classpath, select **Options** from the Collaboration Editor **Tools** menu.

3 In the **Java ClassPaths** window, select the driver files from the list of available files.

4 From the **File** menu in the Collaboration Editor, select **Compile**.

This will enable the .ctl file to find your chosen drivers during run time.

connection method

Description

DataSource.connection_method:

Specifies which method is used to connect to the database server.

URL: A connection is established using the information specified in **jdbc url**.

Pooled Data Source: A connection is established using the information specified in **data source attributes**.

XA Data Source: A connection is established using the information specified in **data source attributes**. If the driver does not support XA, it will be downgraded to Pooled Data Source. For more information regarding XA, please see the ***e*Gate User Guide***.

Required Values

"URL", "Pooled Data Source", or "XA Data Source". The default is "URL".

jdbc url

Description

DataSource.jdbc_url:

This is the JDBC URL necessary to gain access to the database. The URL usually starts with jdbc; followed by <subprotocol> and ends with information for identifying the data source. For a JDBC-ODBC bridge, the subprotocol is odbc:. The information that identifies the rest of the data source is usually:

```
[;<attribute-name>=<attribute-value>].
```

For additional information on this, please consult the documentation of your specific driver.

For SUN's JDBC-ODBC bridge, the URL looks like this:

```
jdbc:odbc:<data-source-name>[;<attribute-name>=<attribute-value>]
```

For example:

```
jdbc:odbc:myDataSource;Cachesize=300
```

If you do not select URL in the **connection method** this parameter is ignored.

Required Values

"URL", "Pooled Data Source", or "XA Data Source". The default is "URL".

data source attribute value pair separator

Description

DataSource.data_source_attribute_value_pair_seperator:

This entry specifies the character separator used to separate the attribute-value pair used **data source attributes**. For example, the attribute-value pair "**ServerName!myHost**" has "**!**"as a separator.

Required Values

Any valid separator. The default is "**!**"

data source attributes

Description

DataSource.data_source_attributes:

A list of "!" separated attribute-value pairs. This information is used to identify the database and set the connection properties. The attribute name should be exactly the same as the one that is specified in the documentation of the driver you selected.

The attribute name should be exactly the same as the one that is specified in the driver documentation and the value should be a valid one. The whole list is used to specify the connection properties. Clear an attribute to disable it.

For example: PortNumber!8888

The separator used in this parameter should match the one specified in **data source attribute value pair separator**

Data source attributes are not used if the **connection method** you specified was URL.

Required Values

Any valid separator. The default separator is "**!**".

user name

Description

Specifies the user name this e*Way uses to connect to the database.

Required Values

Any valid string.

password

Description

Encrypted password:

Specifies the password used to access the database.

Required Values

Any valid string.

timeout

Description

This is the login time out in seconds.

Required Values

The default value is 300 seconds.

3.1.2 Connector Settings

The Connector settings define the high level characteristics of the e*Way Connection.

connector

Description

Connector type:

Specifies the connector type for the JDBC ODBC connection. Currently there is only one type, **DB**.

Required Values

The default is **DB**.

class

Description

Specifies the class name of the JDBC connector object.

Required Values

The default is **com.stc.eways.jdbcx.DbConnector**.

transaction mode

This parameter specifies how a transaction should be handled.

- **Automatic** — e*Gate takes care of transaction control and users should not issue a commit or rollback. If you are using XA, you must set your **connection establishment mode** and your **transaction mode** both to **Automatic**.
- **Manual** — You must manually take care of transaction control by issuing a commit or rollback.

Required Values

The required values are **Automatic** or **Manual**. The default is set to **Automatic**.

Mixing XA-Compliant and XA-Noncompliant e*Way Connections

A Collaboration can be XA-enabled if and only if all its sources and destinations are XA-compliant e*Way Connections. However, XA-related advantages can accrue to a Collaboration that uses one (and only one) e*Way Connection that is transactional but not XA-compliant—in other words, it connects to exactly one external system that supports commit/rollback (and is thus transactional) but does not support two-phase commit (and is thus not XA-compliant). See the *e*Gate Integrator User's Guide* for usage and restrictions.

connection establishment mode

This parameter specifies how a connection with the database server is established and closed.

- **Automatic** indicates that the connection is automatically established when the collaboration is started and keeps the connection alive as needed. If you are using XA, you must set your **connection establishment mode** and your **transaction mode** both to **Automatic**.
- **OnDemand** indicates that the connection is established on demand as business rules requiring a connection to the external system are performed. The connection is closed after the methods are completed.

- **Manual** indicates that the user explicitly calls the connection connect and disconnect methods in their collaboration as business rules.

Required Values

The required values are **Automatic**, **OnDemand,** or **Manual**. The default is set to **Automatic**.

Note: *If you are using **Manual connection establishment mode**, you must also use **Manual transaction mode**.*

connection inactivity timeout

This value is used to specify the timeout for the Automatic connection establishment mode. If this is set to 0, the connection is not brought down due to inactivity. The connection is always kept alive; if it goes down, re-establishing the connection is automatically attempted. If a non-zero value is specified, the connection manager tries to monitor for inactivity so that the connection is brought down if the timeout specified is reached.

Required Values

Any valid string.

connection verification interval

This value is used to specify the minimum period of time between checks for connection status to the database server. If the connection to the server is detected to be down during verification, your collaboration's **onDown** method is called. If the connection comes up from a previous connection error, your collaboration's **onUp** method is called.

Required Values

Any valid string.

3.2 Connection Manager

The Connection Manager allows you to define the connection functionality of your e*Way. You choose:

- When an e*Way connection is made.
- When to close the e*Way connection and disconnect.
- What the status of your e*Way connection is.
- When the connection fails, an **OnConnectionDown** method is called by the Collaboration

The Connection Manager was specifically designed to take full advantage of the e*Gate enhanced functionality. If you are running e*Gate 4.5.1 or earlier, this enhanced functionality is visible but is ignored.

The Connection Manager is controlled in the e*Way configuration as described in **Connector Settings** on page 42. If you choose to manually control the e*Way connections, you may find the following chart helpful.

Figure 7 e*Way Connection Control methods

	Automatic	On-Demand	Manual
onConnectionUp	yes	no	no
onConnectionDown	yes	yes only if the connection attempt fails	no
Automatic Transaction (XA)	yes	no	no
Manual Transaction	yes	no	no
connect	no	no	yes
isConnect	no	no	yes
disconnect	no	no	yes
timeout or connect	yes	yes	no
verify connection interval	yes	no	no

Controlling When a Connection is Made

As a user, you can control when a connection is made. Using Connector Settings, you can choose to have e*Way connections controlled manually — through the Collaboration, or automatically — through the e*Way Connection Configuration. If you choose to control the connection you can specify the following:

- To connect when the Collaboration is loaded.
- To connect when the Collaboration is executed.
- To connect by using an additional connection method in the ETD.
- To connect by overriding any custom values you have assigned in the Collaboration.
- To connect by using the isConnected() method. The isConnected() method is called per connection if your ETD has multiple connections.

Controlling When a Connection is Disconnected

In addition to controlling when a connection is made, you can also manually or automatically control when an e*Way connection is terminated or disconnected. To control the disconnect you can specify:

- To disconnect at the end of a Collaboration.
- To disconnect at the end of the execution of the Collaborations Business Rules.
- To disconnect during a timeout.
- To disconnect after a method call.

Controlling the Connectivity Status

You can control how often the e*Way connection checks to verify if it is still alive and you can set how often it checks. See **Connector Settings** on page 42.

3.3 Monk ODBC Configuration

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

This section 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 Schema Designer's online help.

This section provides information about the following:

3.3.1 Configuration Overview

Before you can run the ODBC e*Way, you must configure it using the e*Way Edit Settings window, which is accessed form the e*Gate Schema Designer GUI. The ODBC e*Way package includes a default configuration file which you can modify using this window.

3.4 e*Way Configuration Parameters

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

To change e*Way configuration parameters

1. In the Schema Designer's Component editor, select the e*Way you want to configure and display its properties.
2. Under **Configuration File**, click **New** to create a new file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file.
3. In the **Additional Command Line Arguments** box, type any additional command line arguments that the e*Way may require, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have a specific need to do so.

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

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

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

3.4.1 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 48** 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 shuts down and exits.

Required Values

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

Forward External Errors

Description

Selects whether error messages that begin with the string **DATAERR** that are received from the external system are queued to the e*Way's configured queue. See **"Exchange Data with External Function" on page 62** for more information.

Required Values

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

See **"Schedule-driven data exchange functions" on page 57** for information about how the e*Way uses this function.

3.4.2 Communication Setup

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

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

Start Exchange Data Schedule

Description

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

Required Values

One of the following:

- 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 would cause an invalid date selection (i.e. the 30th of every month would not include February).

Additional Requirements

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

- **Exchange Data with External Function** on page 62
- **Positive Acknowledgment Function** on page 64
- **Negative Acknowledgment Function** on page 65

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

Additional Information

When the schedule starts, the e*Way determines whether it is waiting to send an ACK or NAK to the external system (using the Positive 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 62**, **"Exchange Data Interval" on page 49**, and **"Stop Exchange Data Schedule" on page 49** 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 numbers of days, be sure not to provide boundaries that would cause an invalid date selection (i.e. the 30th of every month would 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 48** and **"Stop Exchange Data Schedule" on page 49** 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 63** 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 63** 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**.

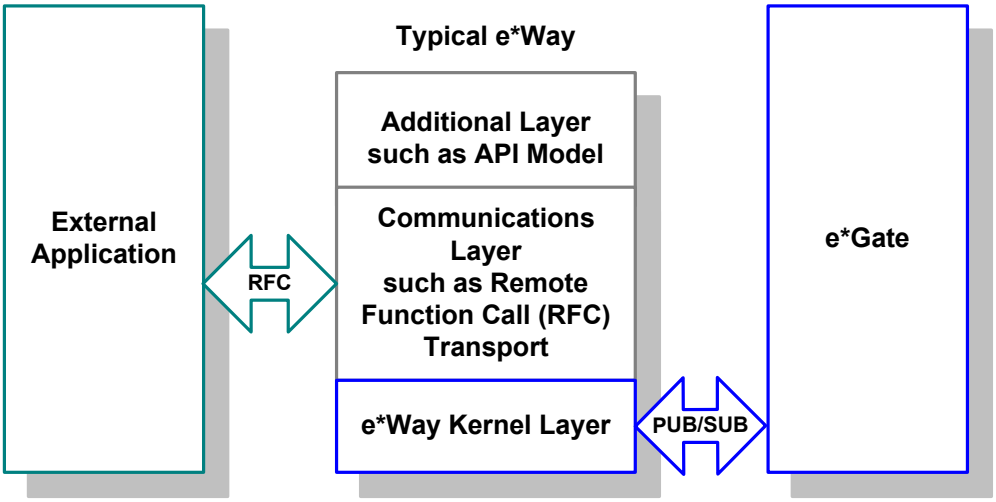
See **"Exchange Data with External Function" on page 62** for more information.

3.4.3 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.

Architecturally, an e*Way can be viewed as a multi-layered structure, consisting of one or more layers that handle communication with the external application, built upon an e*Way Kernel layer that manages the processing of data and subscribing or publishing to other e*Gate components (see Figure 8).

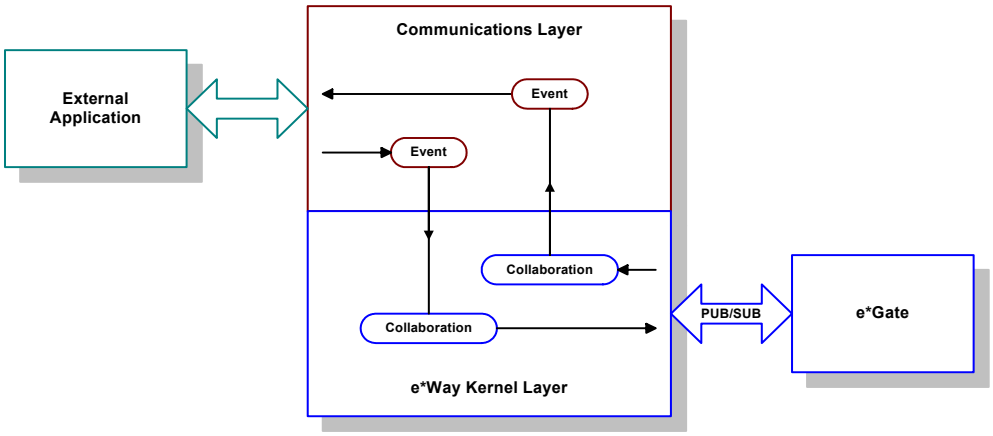
Figure 8 Typical e*Way Architecture



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

The upper layers of the e*Way use Monk functions to start and stop scheduled operations, exchange data with the external system, package data as e*Gate "Events," send those Events to Collaborations, and manage the connection between the e*Way and the external system (see **Figure 9 on page 52**).

Figure 9 Basic e*Way Operations



Configuration options that control the Monk environment and define the Monk functions used to perform these basic e*Way operations are discussed in **Chapter 6**. You can create and modify these functions using the SeeBeyond Collaboration Rules Editor or a text editor (such as Microsoft Wordpad or Notepad).

The upper layers of the e*Way are single-threaded. Functions run serially, and only one function can be executed at a time. The e*Way Kernel 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.

The basic set of e*Way Kernel Monk functions is described in **Chapter 6**. Generally, e*Way Kernel Monk functions should be called directly only when there is a specific need not addressed by higher-level Monk functions, and should be used only by experienced developers.

Basic e*Way Processes

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 61 (also see **Monk Environment Initialization File** on page 60)
Connection	**External Connection Establishment Function** on page 63 **External Connection Verification Function** on page 63 **External Connection Shutdown Function** on page 64

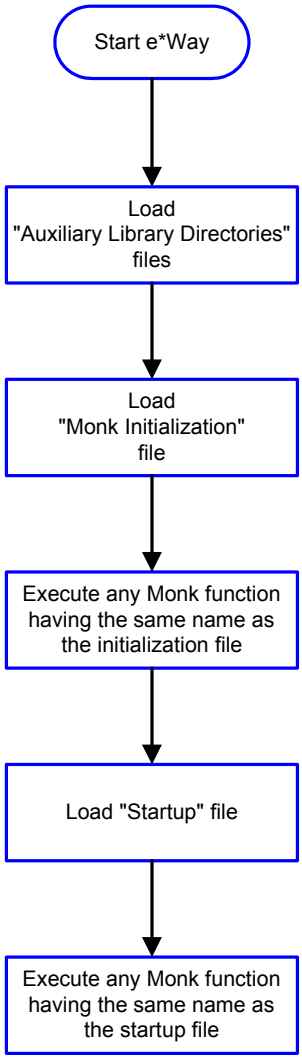
Type of Operation	Name
Schedule-driven data exchange	**Exchange Data with External Function** on page 62 **Positive Acknowledgment Function** on page 64 **Negative Acknowledgment Function** on page 65
Shutdown	**Shutdown Command Notification Function** on page 66
Event-driven data exchange	**Process Outgoing Message Function** on page 61

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

Initialization Functions

Figure 10 illustrates how the e*Way executes its initialization functions.

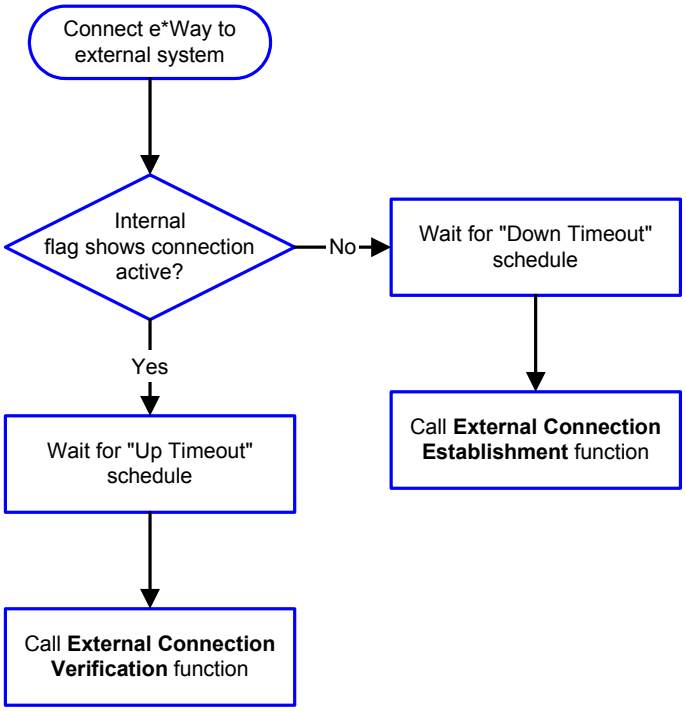
Figure 10 Initialization Functions



Connection Functions

Figure 11 illustrates how the e*Way executes the connection establishment and verification functions.

Figure 11 Connection establishment and verification functions

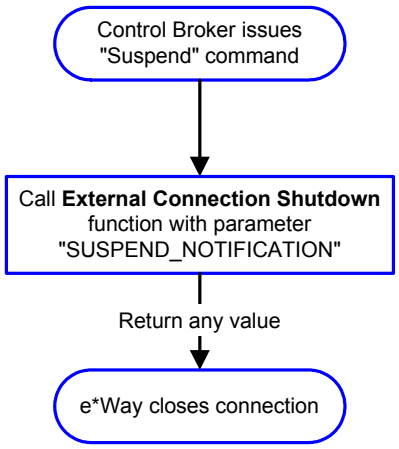


Note: *The e*Way selects the connection function based on an internal "up/down" flag rather than a poll to the external system. See* **Figure 13 on page 57** *and* **Figure 15 on page 59** *for examples of how different functions use this flag.*

User functions can manually set this flag using Monk functions. See **send-external-up** *on page 352 and* **send-external-down** *on page 351 for more information.*

Figure 12 illustrates how the e*Way executes its "connection shutdown" function.

Figure 12 Connection shutdown function



Schedule-driven Data Exchange Functions

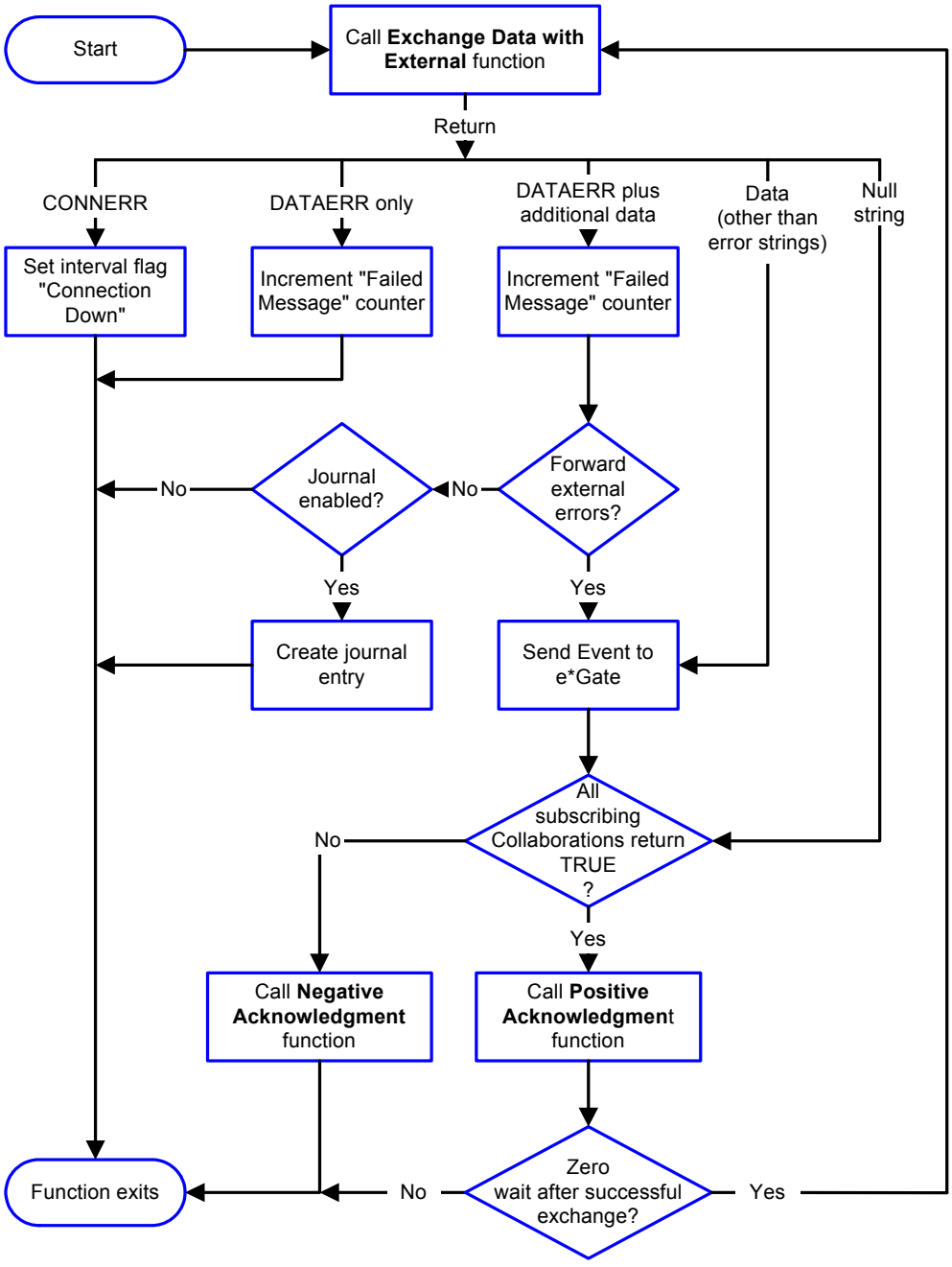
Figure 13 on page 57 illustrates how the e*Way performs schedule-driven data exchange using the **Exchange Data with External Function**. The **Positive Acknowledgement Function** and **Negative Acknowledgement Function** are also called during this process.

"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.

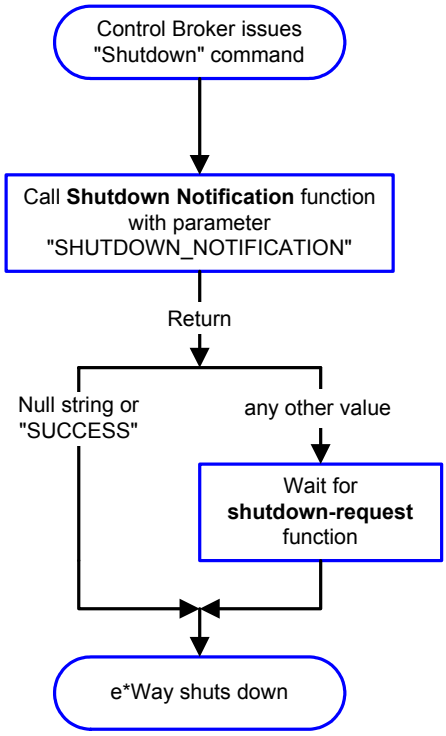
Figure 13 Schedule-driven data exchange functions



Shutdown Functions

Figure 14 illustrates how the e*Way implements the shutdown request function.

Figure 14 Shutdown functions

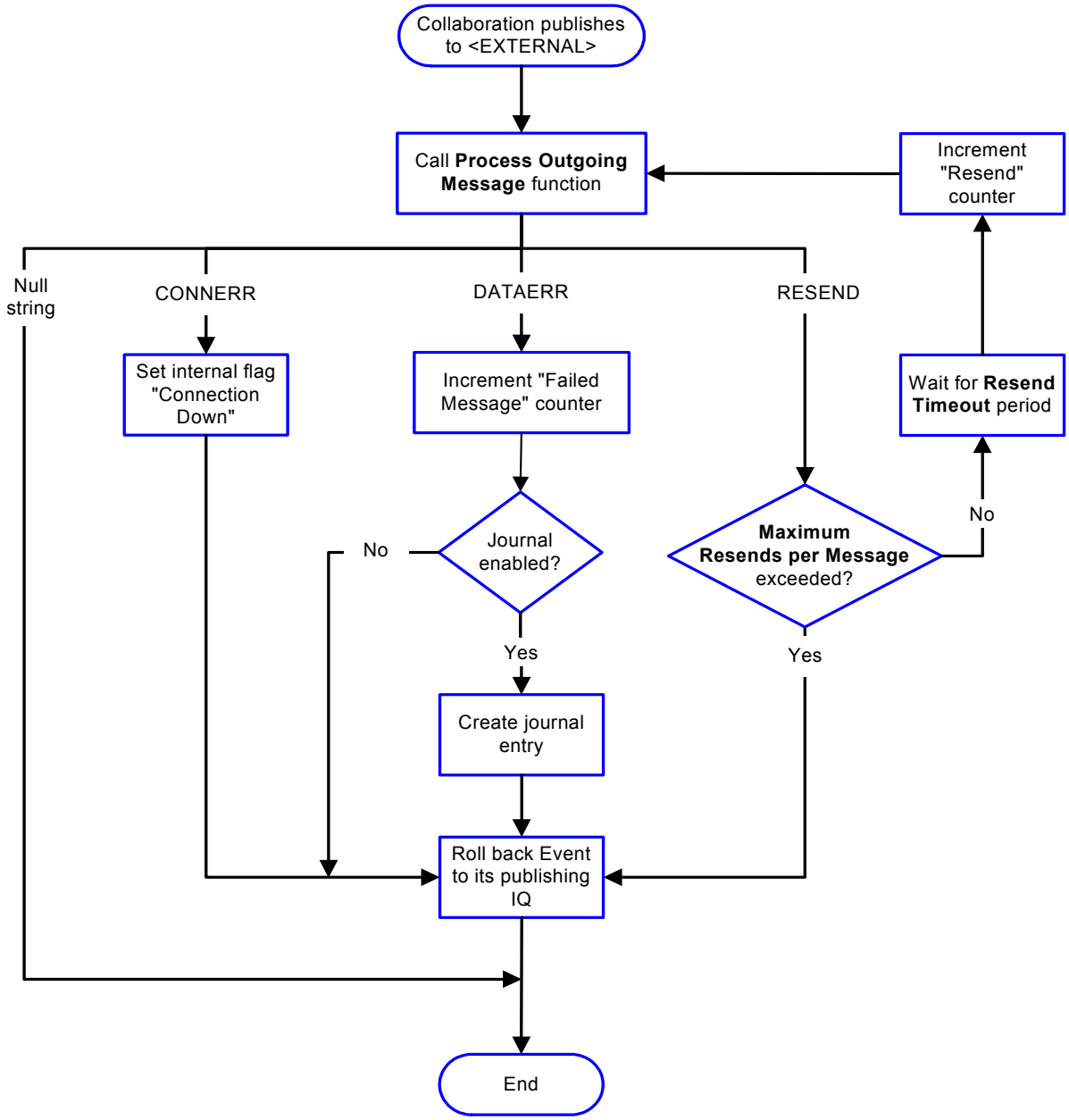


Event-driven Data Exchange Functions

Figure 15 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 15 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 aare 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 60 for more information about the "load path."(The default is **db-stdver-init** on page 364.)

Additional information

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

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

The internal function that loads this file is called once when the e*Way first starts up (see **Figure 10 on page 54**).

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 372.)

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 10 on page 54**). For example, for a file named **my-startup.monk**, the e*Way would attempt to execute the function **my-startup**.

Process Outgoing Message Function

Description

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

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. ***You may not leave this field blank.*** (The default is **db-stdver-proc-outgoing** on page 367 or **db-stdver-proc-outgoing-stub** on page 369**.**)

Additional Information

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

The e*Way invokes this function when one of its Collaborations publishes an Event to an <EXTERNAL> destination (as specified within the Schema Designer). The function returns one of the following (see **Figure 15 on page 59** 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 wish to use* ***event-send-to-egate*** *to enqueue failed Events in a separate IQ, the e*Way must have an inbound Collaboration (with appropriate IQs) configured to process those Events. See* **event-send-to-egate** *on page 349 for more information.*

Exchange Data with External Function

Description

Specifies a Monk function that initiates an exchange of data with an external system.This function can exchange Events either inbound or outbound. This function is used with schedule based exchanges of data, 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 362 or **db-stdver-data-exchg-stub** on page 363.)

Additional Information

The function accepts no input and must return a string (see **Figure 13 on page 57** 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 13 on page 57** 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 354 for more information).

External Connection Establishment Function

Description

Specifies a Monk function that the e*Way calls to establish (or re-establish) a connection to 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 field cannot be left blank***. (The default is **db-stdver-conn-estab** on page 357.)

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 360.)

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 external system.

Required Values

The name of a Monk function. (The default is **db-stdver-conn-shutdown** on page 359.)

Additional Information

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

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

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

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 call when ***all*** the Collaborations to which the e*Way sent data have processed and enqueued that data successfully.

Required Values

The name of a Monk function, or the name of a file (optionally including path information) containing a Monk function. This parameter is required if the **Exchange Data with External** function is defined. (The default is **db-stdver-pos-ack** on page 366.)

Additional Information

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

- "CONNERR": Indicates a problem with the connection to the external system. When the connection is re-established, the Positive Acknowledgment function 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 data 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 365.)

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 371.)

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 will not proceed until the Monk function **shutdown-request** is executed.

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

3.4.4 Database Setup

Database Type

Description

Specifies the type of database.

Required Values

DB2, **ODBC**, **ORACLE7**, **ORACLE8**, **ORACLE8i**, **SYBASE11**, or **SYBASE12**

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

Database Name

Description

The name of the database.

Required Values

None. Any valid string.

User Name

Description

The name used to access the database.

Required Values

None. 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 Schema Designer) 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 Schema Designer's online Help system, or the "stcregutil" command-line utility in the "e*Gate Integrator System Administration and Operations Guide".*

Chapter 4

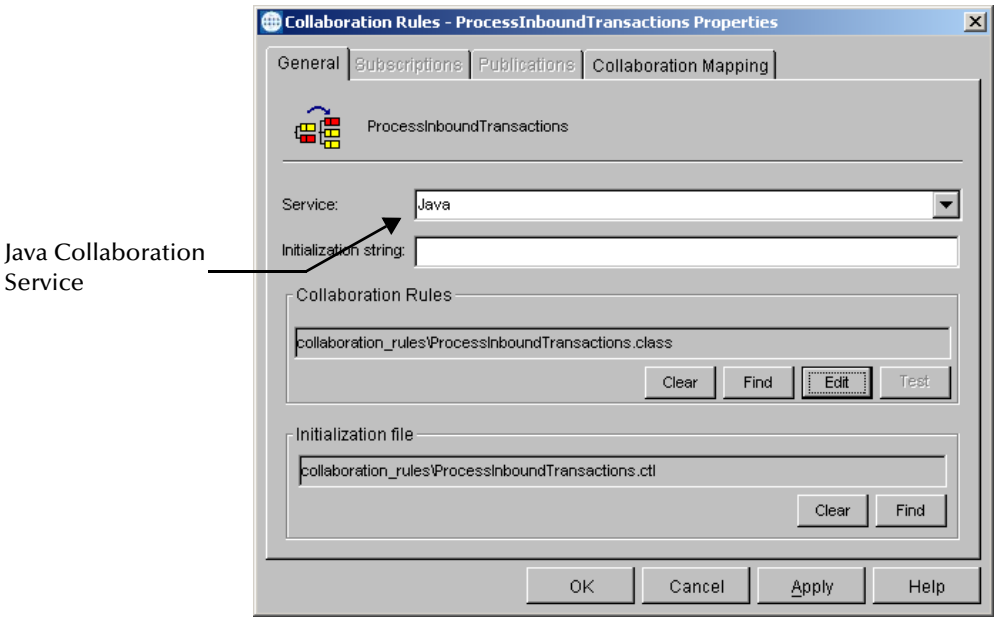
Implementation

This chapter discusses how to implement the JDBC/ODBC and ODBC e*Ways in a production environment. Also included is a sample configuration.

4.1 Implementing Java-enabled Components

An e*Way or a BOB can be Java-enabled by selecting the Java Collaboration Service in the Collaboration Rules Properties (see Figure 16). Either of these components can use e*Way Connections to exchange data with external systems.

Figure 16 The Java Collaboration Service



4.1.1 The Java Collaboration Service

The Java Collaboration Service makes it possible to develop external Collaboration Rules that will execute e*Gate business logic using Java code. Using the Java Collaboration Editor, you create Java classes that utilize the **executeBusinessRules()**, **userTerminate()**, and **userInitialize()** methods.

For more information on the Java Collaboration Service and sub collaborations, see the *e*Gate Integrator Collaboration Services Reference Guide*. For more information on the Java ETD Editor and the Java Collaboration Editor, see the *e*Gate Integrator User's Guide*.

4.1.2 Java-enabled Components

To make an e*Gate component Java-enabled, the component's Collaboration Rule must use the Java Collaboration Service. This requires all the intermediate components to also be configured correctly, since there is not a direct relationship between the e*Way/BOB and the Collaboration Service.

The e*Way/BOB requires one or more Collaborations. The Collaboration uses a Collaboration Rule. The Collaboration Rule uses a Collaboration Service. In order for the e*Way or BOB to be Java-enabled, the component's Collaboration Rule must use the Java Collaboration Service.

4.2 The Java ETD Builder

The Java ETD Builder is used to generate a Java-enabled ETD. The ETD Builder connects to the external database and generates the ETD corresponding to the external tables and procedures.

Note: *Database ETDs are not messagable.*

4.2.1 The Parts of the ETD

There are four possible parts to the Java-enabled Event Type Definition as shown in Figure 17.

Figure 17 The Java-enabled ETD

- **Element** – This is the highest level in the ETD tree. The element is the basic container that holds the other parts of the ETD. The element can contain fields and methods.
- **Field** – Fields are used to represent data. A field can contain data in any of the following formats: string, boolean, int, double, or float.
- **Method** – Method nodes represent actual Java methods.
- **Parameter** – Parameter nodes represent the Java methods' parameters.

4.2.2 Using the DBWizard ETD Builder

The DBWizard ETD Builder generates Java-enabled ETDs by connecting to external data sources and creating corresponding ETDs. The ETD Builder can create ETDs based on any combination of tables, stored procedures, or prepared SQL statements.

Field nodes are added to the ETD based on the tables in the external data source. Java method and parameter nodes are added to provide the appropriate JDBC functionality.

Note: *For more information on the Java methods, refer to your JDBC developer's reference.*

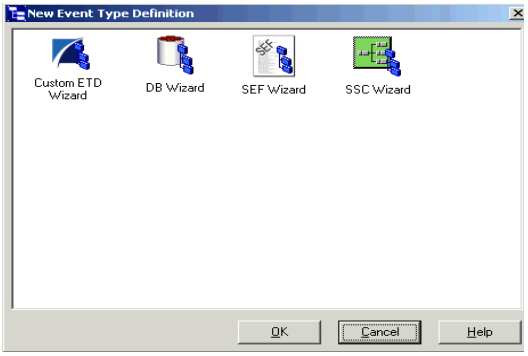
You can use the DBWizard ETD Builder to create four types of ETDs:

- **Tables**
- **Views**
- **Procedures**
- **Prepared statements**

To create a new ETD using the DBWizard

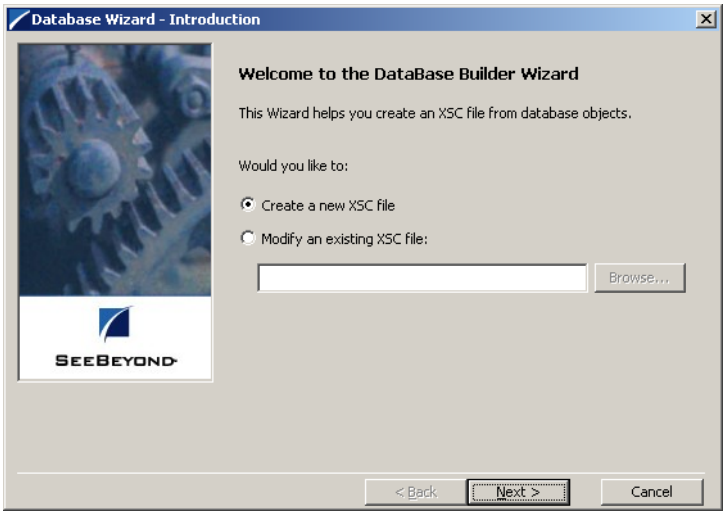
1. From the **Options** menu of the Schema Designer, select **Default Editor**.
2. Verify that **Java** is selected, then click **OK**.
3. Click **ETD Editor** to launch the Java ETD Editor.
4. In the **Java ETD Editor**, click **New** to launch the New Event Type Definition Wizard.
5. In the **New Event Type Definition Wizard** window, select the **DBWizard** and click **OK**.

Figure 18 New Event Type Definition



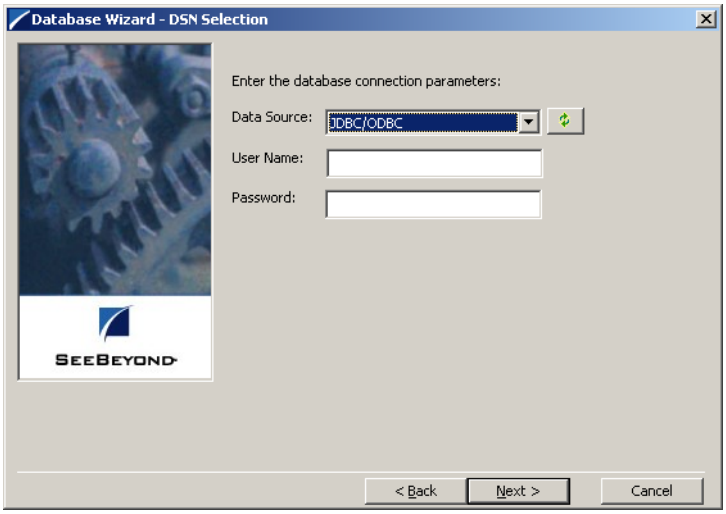
6. In the Database Wizard's **Introduction** window, do one of the following:
   - Enter the name of the new .xsc file you want to create, or
   - Indicate the.xsc file you want to edit by browsing to its location or entering its name.

Figure 19 Database Wizard - Introduction



7 In the **DSN Selection** window, select your **Data Source** from the drop-down list and enter your **User Name** and **Password**.

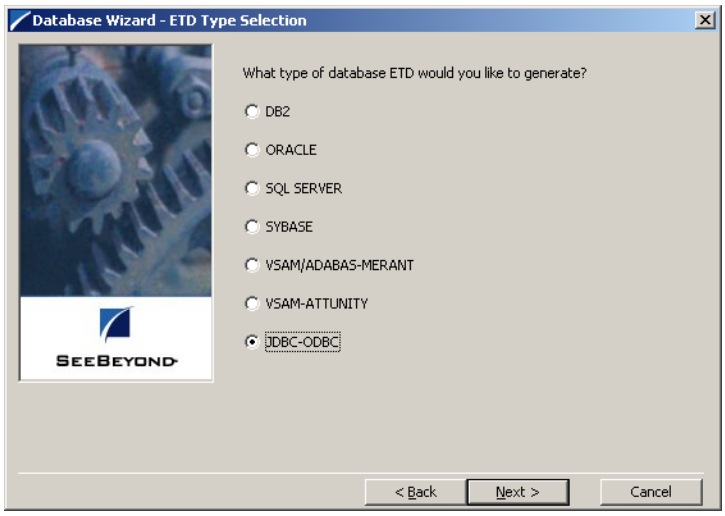
Figure 20 Database Wizard - DSN Selection



8 In the **ETD Type Selection** window, select what type of database ETD you would like to generate. The data source you selected in the **DSN Selection** window is the default.

Note: *Do not change the default unless instructed to do so by SeeBeyond personnel.*

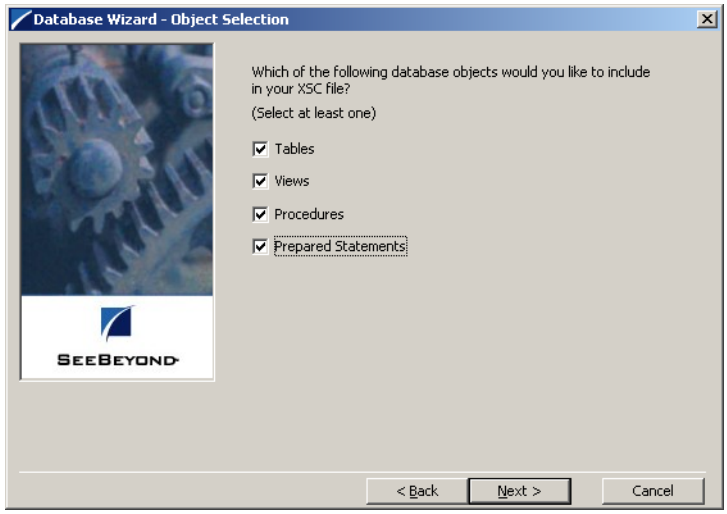
Figure 21 Database Wizard - ETD Type Selection



9 In the **Object Selection** window, select any combination of **Tables**, **Views**, **Procedures**, or **Prepared Statements** you would like to include in your .xsc file.

10 Click **Next** to continue.

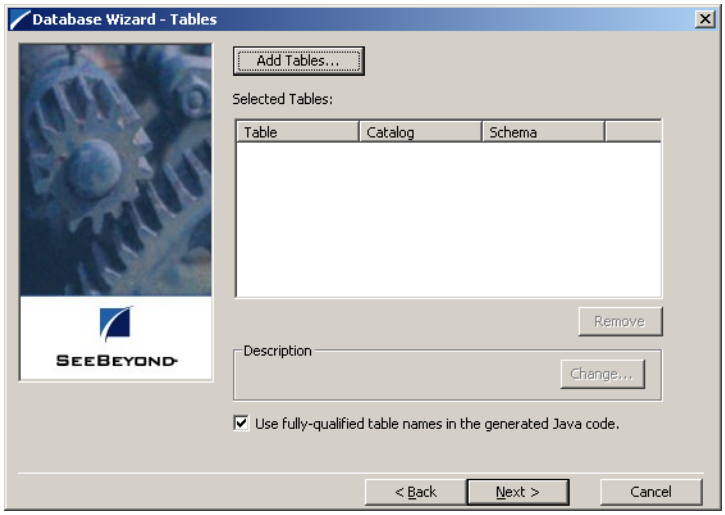
Figure 22 Database Wizard - Object Selection



11 In the **Tables** window, click **Add Tables**.

Note: *You will use the same procedure to add tables as that for adding views and procedures. Detailed instructions for adding tables are provided in the next set of steps, and you will be referred back to these instructions later in the document for details about adding views and procedures.*

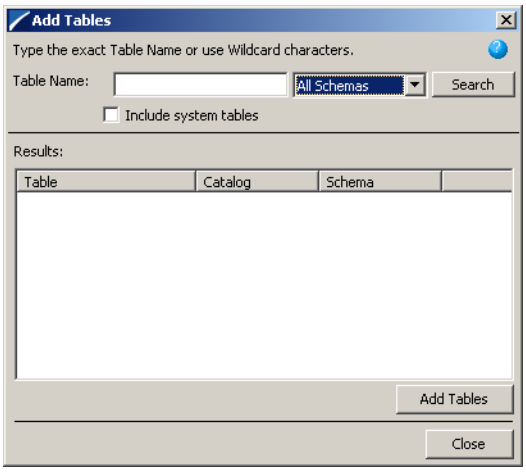
Figure 23 Database Wizard - Tables



12 In the **Add Tables** window, enter the name of the database table. You can enter the exact table name, or you can enter wildcard characters. Select **Include System Tables** if you wish to include them.

13 Click **Search**.

Figure 24 Add Tables



Note: *To see a list of valid wildcard characters, click the question mark icon in the **Add Tables** window.*

14 If your search was successful, you will see a list of tables in the **Results** window. To add tables to your .xsc, double-click each table name, or click the names of the tables and click **Add Tables**.

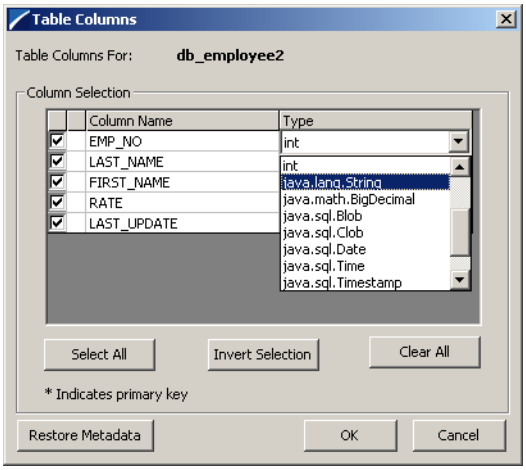
Note: *You may also use adjacent selections or nonadjacent selections to select multiple table names.*

15 When you have finished adding tables, click **Close**.

16 In the **Tables** window of the Database Wizard (see **Figure 23 on page 73**), review the tables you have selected. If you would like to change any of the tables you have selected, click **Change** to view the **Columns Selection** window.

17 In the **Columns Selection** window, you can change your table selection choices. You can also change the data type for each table by highlighting the data type and selecting a different data type from the drop-down list.

Figure 25 Columns Selection



18 Once you have completed your choices in the **Columns Selection** window, click **OK**.

19 In the **Tables** window of the Database Wizard (see **Figure 23 on page 73**), review the tables you have selected.

20 If you do not want to use fully-qualified table names in the generated Java code, clear the check box.

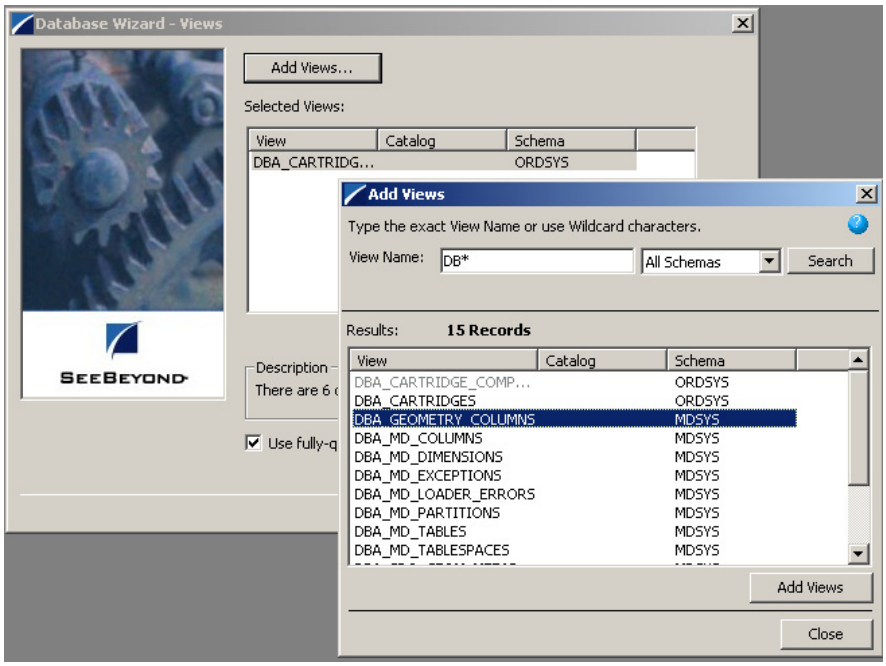
21 When you have finished configuring the settings in the **Tables** window, click **Next** to continue.

22 If you selected **Views** in the **Object Selection** window (**Figure 22 on page 72**), you are presented with the **Views** window.

23 Select and add views to your .xsc file by following the same instructions as those for selecting and adding tables.

Note: **Figure 26 on page 75** *shows the* ***Add Views*** *window. Views are read-only.*

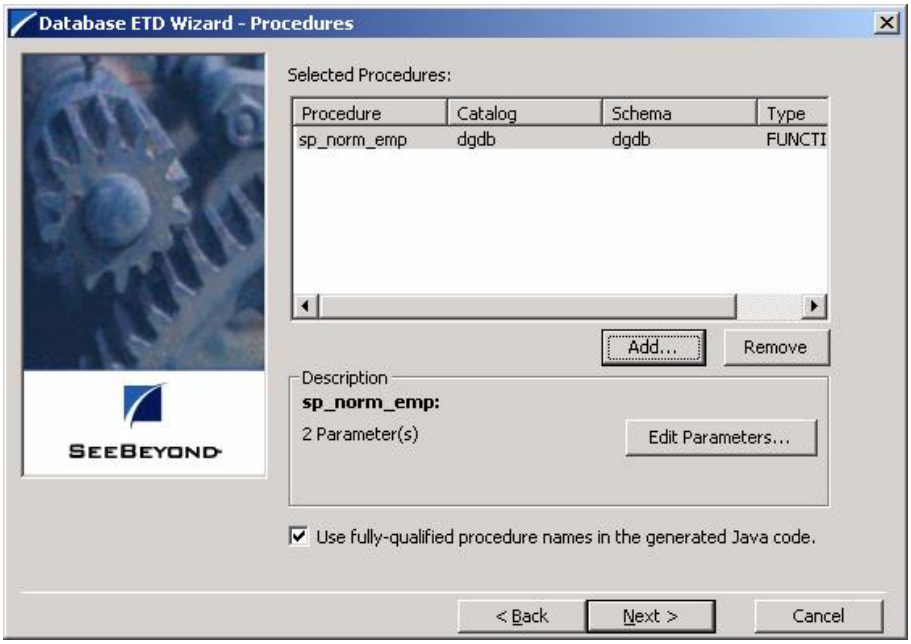
Figure 26 Database Wizard - Views



24 When you have finished configuring the settings in the **Views** window, click **Next** to continue.

25 If you selected **Procedures** in the **Object Selection** window (**Figure 22 on page 72**), you are presented with the **Procedures** window.

Figure 27 Database Wizard - Procedures



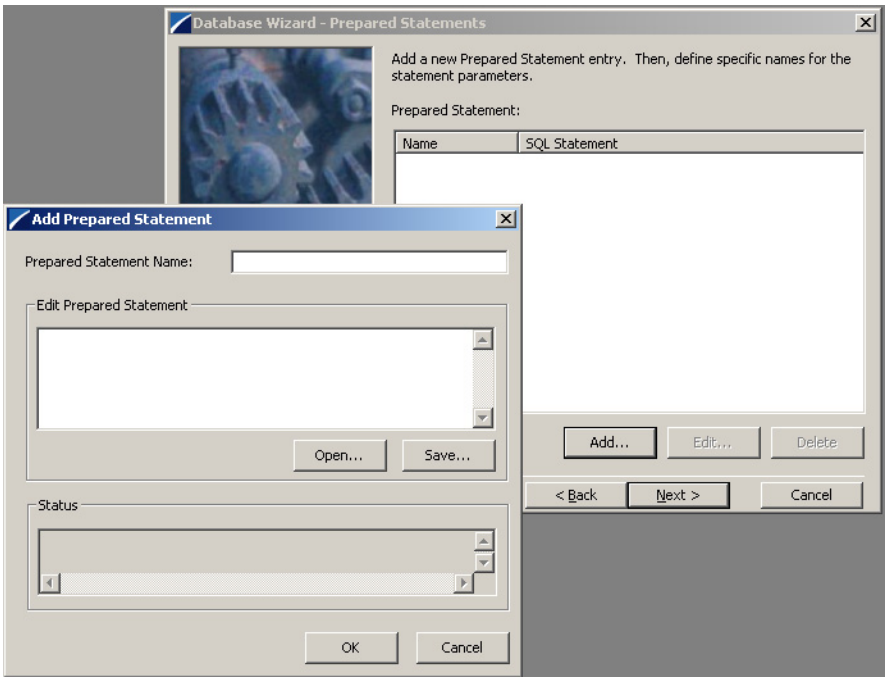
26 Select and add procedures to your .xsc file by following the same instructions as those for selecting and adding tables.

Note: *If you do not want to use fully-qualified procedure names in generated Java code, make sure that you you clear the check box in the* ***Procedures*** *window.*

27 When you have finished configuring the settings in the **Procedures** window, click **Next** to continue.

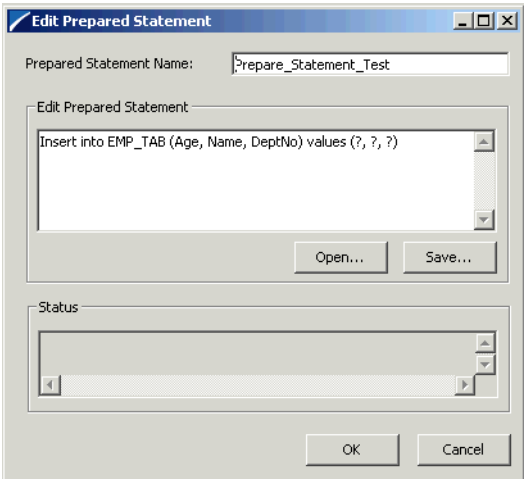
28 If you selected **Prepared Statements** on the **Object Selection** window (**Figure 22 on page 72**), you are presented with the **Prepared Statement** window.

Figure 28 Database Wizard - Prepared Statements



29 Click **Add** to add a new prepared statement

Figure 29 Add Prepared Statement



30 To add **Prepared Statements** to your .xsc file, click **Add** in the **Prepared Statements** window.

31 Enter the **Prepared Statement Name**.

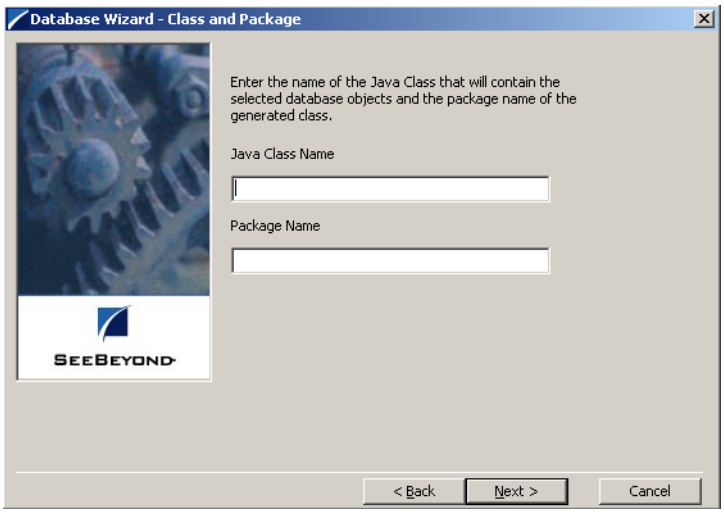
32 Enter a prepared SQL statement.

33 Click Open to open a pre-exisiting statement, or Save to save the current statement.

34 Repeat steps 31-33 above to add additional statements. Click **Next** to continue when you are finished.

35 In the **Class and Package** window, enter the **Java Class Name** that will contain the selected tables and/or procedures and the **Package Name** of the generated classes.

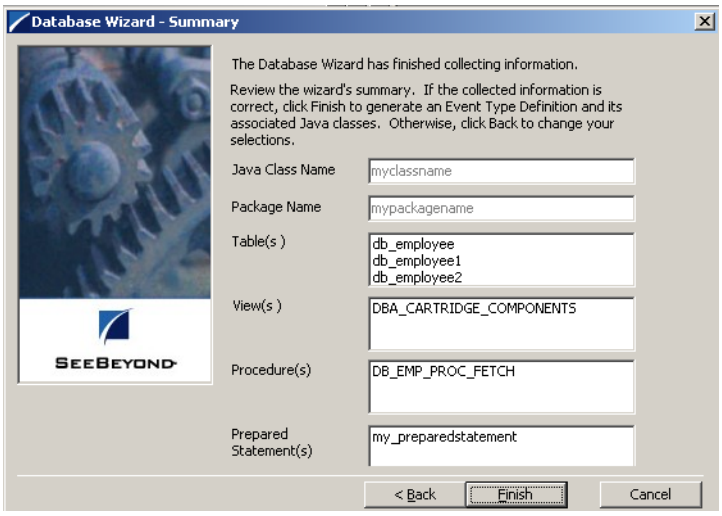
Figure 30 Database Wizard - Class and Package



36 Click **Next** to continue.

37 On the **Summary** screen, view the summary of the database wizard information.

Figure 31 Database Wizard - Summary



38 Click **Finish** to begin generating the ETD.

The Generated ETDs

As the preceding instructions illustrate, the Database Wizard ETD builder can create three editable ETDs and one non-editable ETD. These types of ETDs can be combined with each other.

The four types of ETDs are:

- **The Table ETD** – The table ETD contains fields for each of the columns in the selected table as well as the methods required to exchange data with the external data source. To edit this type of ETD, you will need to open the .xsc in the Database Wizard.
- **The View ETD** - The view ETD contains selected columns from selected tables. View ETD's are read-only.
- **The Stored Procedure ETD** – The stored procedure ETD contains fields which correspond to the input and output fields in the procedure. To edit this type of ETD, you will need to open the .xsc in the Database Wizard.
- **The Prepared Statement ETD** – The prepared statement ETD contains a result set for the prepared statement. To edit this type of ETD, you will need to open the .xsc in the Database Wizard.

4.2.3 Editing an Existing .xsc File Using the Database Wizard

To edit an existing .xsc file that you have created using the Database Wizard, do the following:

1 From the **Options** menu of the Schema Designer, choose **Default Editor**.

2 Verify that **Java** is selected, then click **OK**.

3 From the **Tools** menu, click **ETD Editor...**

4 From the ETD Tool menu click **File**, then click **New**.

5 From the **New Event Type Definition** window, select **DBWizard** and click **OK**.

6 In the **Introduction** window of the Database Wizard, select **Modify an existing XSC file.**

7 Browse to the .xbs file that you would like to edit.

You are now able to edit your .xsc file.

.xsc File Editing Considerations

When editing an .xsc file, make sure that you consider the following:

- When you add a new element type to your existing .xsc, you must reselect any pre-existing elements or you will lose them when the new .xsc is created.
- If you attempt to edit an .xsc that no longer has any exisiting elements in the database, you will see a warning and the element will be dropped from the ETD.

4.3 Using ETDs with Tables, Views, Stored Procedure, and Prepared Statements

Tables, Views, Stored Procedures and Prepared Statements are manipulated through ETDs. Common operations, which are driver dependent, include insert, delete, update, and query.

Note: *Sun's JDBC/ODBC bridge (which is a type 1 driver) does not support these operations.*

4.3.1 Tables

A table ETD represents a database table. It consists of fields and methods. Fields correspond to the columns of a table, while methods are the operations that you can apply to the ETD.

This allows you to perform query, update, insert, and delete SQL operations in a table.

Using the select() method, you can specify the following types of ResultSets:

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

You can also specify ResultSets with a type of concurrency:

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

To perform the update, insert or delete operation, the type of the ResultSet returned by the select() method must be CONCUR_UPDATABLE. Instead of specifying the type of ResultSet and concurrency in the select() method, you can also use the following methods:

- SetConcurrencytoUpdatabale
- SetConcurrentlytoRead Only
- SetScrollTypetoForwardOnly
- SetScrollTypetoScrollSensitive
- SetScrollTypetoInsensitive

The methods should be called before executing the select() method. For example,

```
getDBEmp().setConcurToUpdatable();
getDBEmp().setScroll_TypeToForwardOnly();
getDBEmp().getDB_EMPLOYEE().select("");
```

The Query Operation

To perform a query operation on a table

1. Execute the select() method, with the "where" clause specified if necessary.
2. Loop through the ResultSet using the "next" method.
3. For each loop, process the return record.

For example:

```
getDBEmp().getDB_EMPLOYEE().select("");
While(getDBEmp().getDB_EMPLOYEE().next());
{ //Process the returning record
    getGenericOut.SetPayLoad(getDBEmp().getDB_Employee().
    getDBEmp().getFirstName());
}
```

If you want to check if the last value read was SQL NULL or not, you can use the wasNull() method. It is most useful for native datatypes like "int". Note that a getxxx method should be called before wasNull() is called.

For example:

```
int empNo = getDBEmp().getDB_EMPLOYEE().getEMP_NO();
if (getDBEMP().getDB_EMLOYEE().wasNULL())
{ //Check to see if empNo is SQL NULL
 //Do something if empNo is SQL NULL
}
else
{ //Do something if empNo is not SQL NULL
}
```

If you are using the Merant DataDirect 2.2 drivers in XA mode, you need to set your datasource attribute **SelectMethod** to cursor.

The Insert Operation

To perform an insert operation on a table

1 Execute the select() method. You can specify the following types of ResultSets:

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

You must specify ResultSets with:

- CONCUR_UPDATABLE

2 Move to the insert row by the moveToInsertRow method.

3 Set the fields of the table ETD

4 Insert the row by calling insertRow

This example inserts an employee record.

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().moveToInsertRow();
getDBEmp().getDB_EMPLOYEE().setEMP_NO(123);
. . .
getDBEmp().getDB_EMPLOYEE().setRATE(123.45);
getDBEmp().getDB_EMPLOYEE().insertRow();
```

Figure 32 Insert Method Business Rule

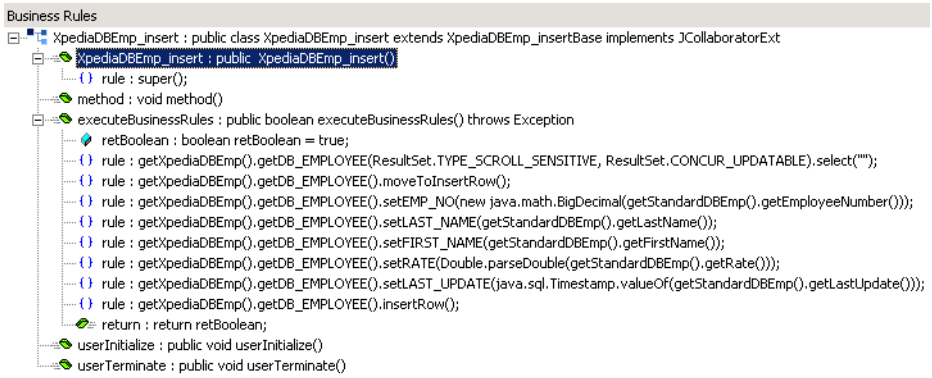


Figure 33 Insert Method Properties

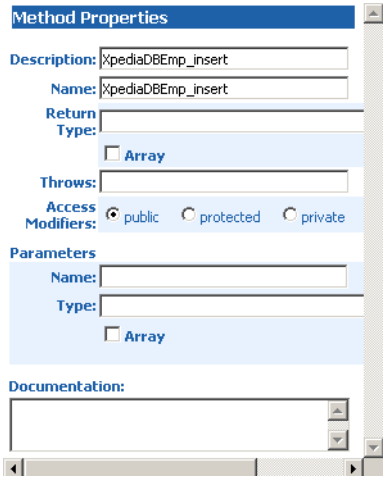


Table ResultSet Behavior

The following procedure will allow you to make repeated insertions using a "select" into the table ResultSet without having to re-populate all the column values.

Before the schema runs, we have

```
SQL> select * from MARKET_TEMP;
```

Where:

```
C1 C2       C3
-- -------- --------
1  A1       B1
```

After the schema runs we have:

```
SQL> select * from MARKET_TEMP;
```

Becomes:

```
C1 C2        C3
--  -------- --------
1   A1       B1
2   A2       B1
3   A3       B1
```

Buffer the value of the selected column by :

```
String buf3 = getTempTbl().getMARKET_TEMP().getC3();
```

Call moveToInsertRow()

```
getTempTbl().getMARKET_TEMP().moveToInsertRow();
```

Set all the columns the first time

```
getTempTbl().getMARKET_TEMP().setC1("2");;
getTempTbl().getMARKET_TEMP().setC2("A2");
getTempTbl().getMARKET_TEMP().setC3(buf3);
```

Call insertRow()

```
getTempTbl().getMARKET_TEMP().insertRow();
```

Set all the columns except the unchanged column.

```
getTempTbl().getMARKET_TEMP().setC1("3");
getTempTbl().getMARKET_TEMP().setC2("A3");
```

Call insertRow()

```
getTempTbl().getMARKET_TEMP().insertRow();
```

In the above example, column C3 always has the same value (buf3).

The Update Operation

To perform an update operation on a table

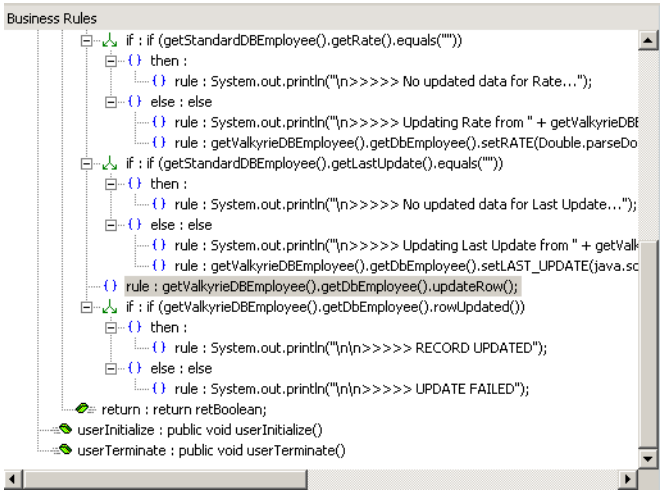
1. Execute the select() method. You can specify the following types of ResultSets:
   - TYPE_FORWARD_ONLY
   - TYPE_SCROLL_INSENSITIVE
   - TYPE_SCROLL_SENSITIVE

   You must specify ResultSets with:
   - CONCUR_UPDATABLE
2. Move to the row that you want to update.
3. Set the fields of the table ETD
4. Update the row by calling **updateRow**.

   In this example, we move to the third record and update the EMP_NO and RATE fields.

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().absolute(3);
getDBEmp().getDB_EMPLOYEE().setEMP_NO(123);
getDBEmp().getDB_EMPLOYEE().setRATE(123.45);
getDBEmp().getDB_EMPLOYEE().updateRow();
```

Figure 34 Update() Method Business Rule



The Delete Operation

To perform a delete operation on a table

1. Execute the select() method. You can specify the following types of ResultSets:
   - TYPE_FORWARD_ONLY
   - TYPE_SCROLL_INSENSITIVE
   - TYPE_SCROLL_SENSITIVE

   You must specify ResultSets with:
   - CONCUR_UPDATABLE
2. Move to the row that you want to delete.
3. Set the fields of the table ETD
4. Delete the row by calling **deleteRow**.

   In this example DELETE the first record of the result set.

```
getDBEmp().getDB_EMPLOYEE(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE).select("");
getDBEmp().getDB_EMPLOYEE().first();
getDBEmp().getDB_EMPLOYEE().deleteRow();
```

4.3.2 The View

Views are used to look at data from selected columns within selected tables. Views are read-only.

For query operations, please refer to "Tables" sub section.

4.3.3 The Stored Procedure

A Stored Procedure ETD represents a database stored procedure. Fields correspond to the arguments of a stored procedure while methods are the operations that you can apply to the ETD. It allows you to execute a stored procedure. Remember that while in the Collaboration Editor you can drag and drop nods from the ETD's into the Collaboration Editor.

Executing Stored Procedures

Assuming that you have the following procedure:

```
create procedure LookupGlobal
( inlocalID in varchar, outglobalProductID out varchar )
as
begin
select outglobalProductID into globalProductID from SimpleLookup
where localID = inlocalID
end
```

The ETD represents the Stored Procedure "LookUpGlobal" with two parameters, an inbound parameter (INLOCALID) and an outbound parameter (OUTGLOBALPRODUCTID) can be generated by the DB Wizard. Representing these as nodes in an ETD allows you to drag values from other ETD's to the input parameters, execute the call, and collect the output parameter data by dragging from it's node to elsewhere.

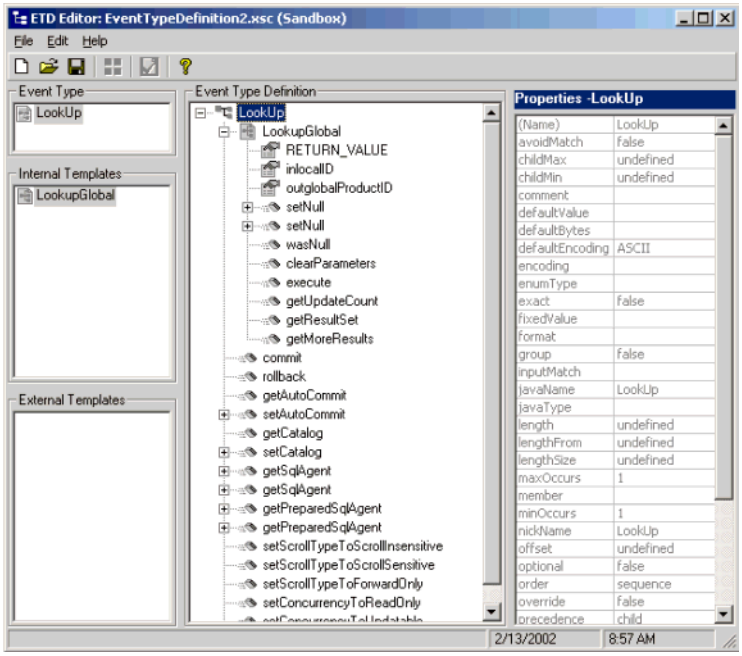
To execute the Stored Procedure

1. Specify the input values.
2. Execute the Stored Procedure.
3. Retrieve the output parameters if any.

For example:

```
getLookUp().getLookUpGlobal().setIntlocalID("123");
getLookUp().getLookUPGlobal().execute();
String s =
getLookUp().getLookUpGlobal.getOutGlobelProductID;
```

Figure 35 Stored Procedure LookUpGlobal



Returning Result Sets from Rows in the Stored Procedure

The following Result Set processing returns rows into the body of the Stored Procedure. Support for this feature depends on the drivers and the specific database you have selected.

A stored procedure in a database can perform update and select operations. The application checks what operation has been performed. The following methods allow you to collect this information:

getUpdateCount:

This method should be called only after a call to **executeRetVal** or g**etMoreResults**, and should be called only once per result. An int greater than 0 represents the number of rows affected by an operation. A value of 0 means no rows were affected or the operation is a DDL command. -1 means the result is a result set or there are no more results.

getResultSet:

When the **executeRetVal** method has been used to execute the stored procedure. This method returns the current result set if there is one. Otherwise, it returns a null value which indicates the result is an update count or there are no more results.

getMoreResults:

Moves to the next result. This method returns true if it gets the next result set. It returns false if it is an update count or there are no more results.

This example prints out the update count and result sets that are returned from the stored procedures.

```
int iRow;
          com.stc.eways.jdbcx.ResultSetAgent rs;
          getpoller().getMrsSp().execute();

          do
          {
             iRow = getpoller().getMrsSp().getUpdateCount();
             System.out.println("\n*********iRow = " +
Integer.toString(iRow));

             if (iRow >= 0)
             {
                getblobout().setField1("Number of rows affected: " +
Integer.toString(iRow));
                getblobout().send();
             }
             else
             {
                rs = getpoller().getMrsSp().getResultSet();
                while (rs.next())
                {
                   getblobout().setField1("Col5 = " +
Integer.toString(rs.getInt(1)) + ", Col6 = " +
Integer.toString(rs.getInt(2)) );
                   getblobout().send();
                }
             }
          }
          while (getpoller().getMrsSp().getMoreResults());
```

Use an outbound parameter to enrich another ETD.

4.3.4 Prepared Statement

A Prepared Statement ETD represents a SQL statement that has been compiled. Fields in the ETD correspond to the input values that users need to provide.

Prepared Statements can be used to perform insert, update, delete, and query operations. A Prepared Statement uses a question mark (?) as a place holder for input. For example:

```
insert into EMP_TAB(Age, Name, Dept No) value(?, ?, ?)
```

To execute a Prepared Statement, set the input parameters and call **executeUpdate()** and specify the input values if any.

```
getPrepStatement().getPreparedStatementTest().setAge(23);
getPrepStatement().getPreparedStatementTest().setName("Peter Pan");
getPrepStatement().getPreparedStatementTest().setDeptNo(6);
getPrepStatement().getPreparedStatementTest().executeUpdate();
```

4.3.5 Batch Operations

While the Java API used by SeeBeyond does not support traditional bulk insert or update operations, there is an equivalent feature that can achieve comparable results, with better performance. This is the "Add Batch" capability. The only modification required is to include the **addBatch()** method for each SQL operation and then the

executeBatch() call to submit the batch to the database server. Batch operations apply only to Prepared Statements.

```
getPrepStatement().getPreparedStatementTest().setAge(23);
getPrepStatement().getPreparedStatementTest().setName("Peter Pan");
getPrepStatement().getPreparedStatementTest().setDeptNo(6);
getPrepStatement().getPreparedStatementTest().addBatch();

getPrepStatement().getPreparedStatementTest().setAge(45);
getPrepStatement().getPreparedStatementTest().setName("Harrison
Ford");
getPrepStatement().getPreparedStatementTest().setDeptNo(7);
getPrepStatement().getPreparedStatementTest().addBatch();
getPrepStatement().getPreparedStatementTest().executeBatch();
```

4.3.6 Database Configuration Node

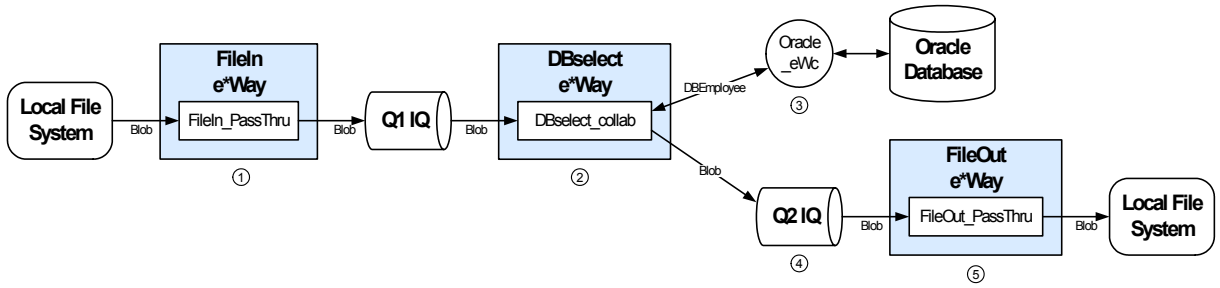
The Database Configuration node allows you to manage the "transaction mode" through the Collaboration if you have set the mode to manual in the e*Way connection configuration. See **"Connector Settings" on page 42**.

4.4 Sample Scenario—Polling from a JDBC/ODBC Generic Database

This section describes how to use the JDBC/ODBC e*Way in a sample implementation. This sample schema demonstrates the polling of records from a generic database and converting the records into e*Gate Events.

Figure 36 shows a graphical overview of the sample schema.

Figure 36 The Database Select Scenario—Overview



1. The **FileIn** e*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **Q1** IQ.
2. The **DBselect** e*Way retrieves the Generic Event (**Blob**) from the IQ. This triggers the rest of the Collaboration which has two parts.
3. The information in **Blob** is used to retrieve information from the database via the **JDBCODBC_eWc** e*Way Connection. This e*Way Connection contains information used by the Collaboration to connect to the JDBCODBC database.

4 The information retrieved from the database is copied to the Generic Event (**Blob**) and published to the **Q2** IQ.

5 The **FileOut** e*Way retrieves the Generic Event (**Blob**) from the **Q2** IQ then writes it out to a text file on the local file system.

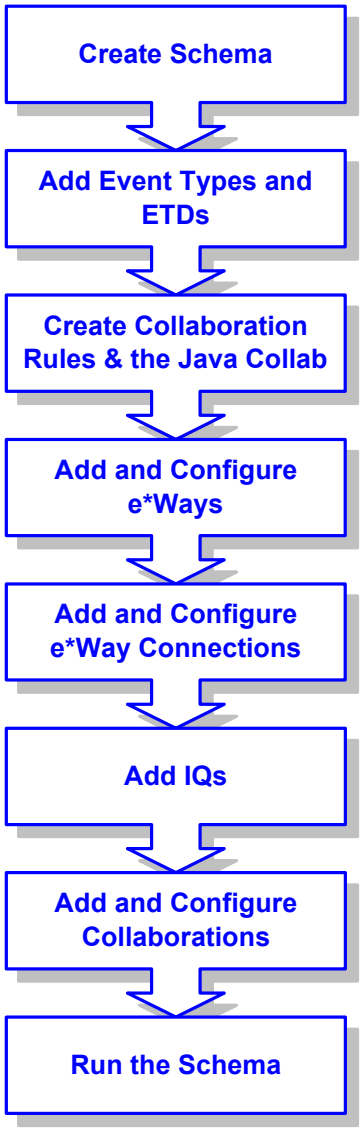
Overview of Steps

The sample implementation follows these general steps:

- **"Create the Schema" on page 91**
- **"Add the Event Types and Event Type Definitions" on page 91**
- **"Create the Collaboration Rules and the Java Collaboration" on page 94**
- **"Add and Configure the e*Ways" on page 98**
- **"Add and Configure the e*Way Connections" on page 100**
- **"Add the IQs" on page 100**
- **"Add and Configure the Collaborations" on page 101**
- **"Run the Schema" on page 102**

Figure 37 provides an illustration of these steps.

Figure 37 Schema Configuration Steps



External Database Tables

The sample uses a simple external JDBCODBC database with a table called **DB_EMPLOYEE**. The table contains the columns shown in Table 4.

Table 4 The DB_EMPLOYEE Table

Column	Format	Description
EMP_NO	INTEGER	The employee number.
LAST_NAME	VARCHAR2	The employee's last name.
FIRST_NAME	VARCHAR2	The employee's first name.
RATE	FLOAT	The employee's pay rate.
LAST_DATE	DATE	The last transaction date for the employee

4.4.1 Create the Schema

The first step in deploying the sample implementation is to create a new schema. After installing the JDBC/ODBC e*Way, do the following:

1. Launch the e*Gate Schema Designer GUI.
2. Log into the appropriate Registry Host.
3. Click **New** to create a new schema.
4. For this sample implementation, enter the name **DBSelect** and click **Open**.

   The Schema Designer launches, displaying the newly the created schema.

4.4.2 Add the Event Types and Event Type Definitions

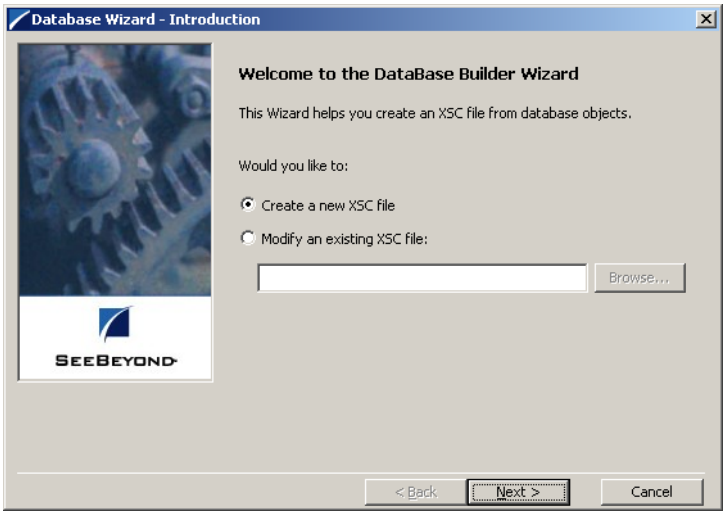
Two Event Types and Event Type Definitions are used in this sample.

- **DBEmployee** – This Event Type represents the layout of the employee records in the **DB_Employee** table. The Event Type uses the **DBEmployee.xsc** Event Type Definition. The ETD is generated by using the Java ETD Editor's Database Wizard (DBWizard).
- **GenericBlob** – This Event Type is used to pass records with no specific format (blob). The Event Type uses the **GenericBlob.xsc** ETD. The ETD is manually created as a fixed-length ETD.

To create the DBEmployee Event Type and ETD

1. From the **Options** menu of the Schema Designer, choose **Default Editor…**.
2. Verify that **Java** is selected, then click **OK**.
3. In the **Components** pane of the Schema Designer, select the **Event Types** folder.
4. Click the **New Event Type** button to add a new Event Type.
5. Enter the name **DBEmployee** and click **OK**.
6. Double-click the new **DBEmployee** Event Type to display its properties.
7. Click the **New** button to create a new Event Type Definition (ETD). The Java Event Type Definition Editor appears.
8. From the **File** menu, choose **New**. The New Event Type Definition dialog box appears.
9. In the **New Event Type Definition** dialog box, select **DBWizard** and click **OK**.
10. In the Database Wizard's **Introduction** window, select **Create a new .XSC file**.

Figure 38 Database Wizard Introduction



11 Click **Next** to continue.

12 Select the **Data Source** from the drop-down list of ODBC data sources.

13 Enter the **User Name** and **Password** used to log into the database.

14 Click **Next** to continue.

15 The **ETD Type Selection** window appears. The DNS source you selected on the previous window is the default selection for this window.

Note: *Do not change this selection type unless instructed to do so by SeeBeyond support personal.*

16 Click **Next** to continue.

17 As this scenario uses a table rather than a procedure, select **Table**.

18 Click **Next** to continue.

19 In the **Tables** window, click **Add Tables**.

20 Enter the exact **Table Name** or enter any valid wildcards.

21 From the drop-down list select the appropriate database schema and click **Search**.

The wizard connects to the data source and display a list of tables.

22 Select the table to be included in the ETD and click **Next**.

The **Java Class Name/ Package Name** window appears.

23 Enter your database name as the **Java Class Name**.

24 Enter **DBEmployee** for the **Package Name**.

25 Click **Next** to continue.

26 Click **Finish** to complete the Wizard. The Wizard will generate and display the ETD.

27 From the **File** menu, choose **Save**.

28 Name the ETD **DBEmployee.xsc** and click **OK**.

29 From the **File** menu, choose **Promote to Run Time** and click **OK** when finished.

30 From the **File** menu, choose **Close** to exit the ETD Editor.

31 In the Event Type properties dialog box, click **OK** to save and close the Event Type.

To create the GenericBlob Event Type and ETD

1 In the **Components** pane of the Schema Designer, select the **Event Types** folder.

2 Click the **New Event Type** button to add a new Event Type.

3 Enter **GenericBlob** and click **OK**.

4 Double-click the new **GenericBlob** Event Type to display its properties.

5 Click the **New** button to create a new Event Type Definition.

The Java Event Type Definition Editor appears.

6 From the **File** menu, choose **New**.

The New **Event Type Definition** window appears.

7 In the **New Event Type Definition** window, select **Standard ETD** and click **OK**.

8 Read the information on the introductory screen, and click **Next** to continue.

The **Root Node Name** / The **Package Name** window will appear.

9 Enter a Root Node Name for the GenericBlob.

10 Enter **GenericBlobPackage** for the **Package Name** and click **Next** to continue.

11 Read the summary information and click **Finish** to generate the ETD.

12 In the **Event Type Definition** pane, right-click the root node, point to **Add Field** in the shortcut menu, and click **As Child Node**.

13 Enter the properties for the two nodes as shown in Table 5.

Table 5 GenericBlob ETD Properties

Node	Property	Value
Root Node	Name	GenericBlob
	Structure	fixed
	Length	undefined
Child Node	Name	Data
	Structure	fixed
	Length	undefined

14 From the **File** menu, choose **Save**.

15 Enter the name **GenericBlob.xsc** and click **OK**.

16 From the **File** menu, choose **Compile And Save**.

17 From the **File** menu, choose **Promote to Run Time**.

18 Click **OK**.

19 From the **File** menu, choose **Close** to exit the ETD Editor.

20 In the Event Type properties dialog box, click **OK** to save and close the Event Type.

4.4.3 Create the Collaboration Rules and the Java Collaboration

The sample scenario uses two Collaboration Rules and one Java Collaboration:

- **GenericPassThru** – This Collaboration Rule is used to pass the GenericBlob Event Type through the schema without modifying the Event.
- **DBSelect** – This Collaboration Rule is used to convert the inbound Event's selection criteria into a SQL statement, poll the external database, and return the matching records as an outbound Event.
- **DBSelectCollab** – This Java Collaboration contains the logic required to communicate with the external database.

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 Schema Designer toolbar, click **Options**.

2 Click **Default Editor...**

3 Select **Java**.

4 Click **OK**.

To create the GenericPassThru Event Type

1 In the components pane of the Schema Designer, select the **Collaboration Rules** folder.

2 Click the **New Collaboration Rules** button to add a new Collaboration Rule.

3 Name the Collaboration Rule **GenericPassThru** and click **OK**.

4 Click the **Properties** button to display the Collaboration Rule's properties.

5 On the **General tab**, select **Pass Through** from the Services drop-down list.

6 Click the **Subscriptions** tab, select the **GenericBlob** Event Type, and click the right arrow.

7 Click the **Publications** tab, select the **GenericBlob** Event Type, and click the right arrow.

8 Click **OK** to save the Collaboration Rule.

To create the DBSelect Event Type

1 In the components pane of the Schema Designer, select the **Collaboration Rules** folder.

2 Click the **New Collaboration Rules** button to add a new Collaboration Rule.

3 Name the Collaboration Rule **DBSelect** and click **OK**.

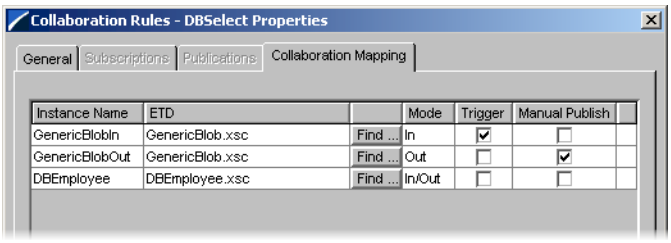
4 Click the **Properties** button to display the Collaboration Rule's properties.

5 In the **Service** list, click **Java**.

6 Click the **Collaboration Mapping** tab.

7 Add three instances. See Figure 39

Figure 39 DBSelect Instances



8 Click **Apply** to save the current changes.

9 Click the **General** tab.

10 Click **New** to create the new Collaboration file.

The Java Collaboration Editor appears. Note that Source and Destination Events are already supplied based on the Collaboration Rule's Collaboration Mapping. See Figure 39.

11 From the **View** menu, choose **Display Code**.

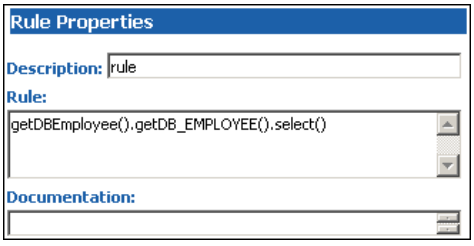
This displays the Java code associated with each of the Collaboration's rules.

12 In the Business Rules pane, select the **retBoolean** rule and click the **rule** button to add a new Rule.

13 In the **Destination Events** pane, expand the **DBEmployee** Event Type until the **select** method is visible.

14 Drag the **select** method into the **Rule** field of the **Rule Properties** pane. Click **OK** to close the dialog box without entering any criteria. See Figure 40.

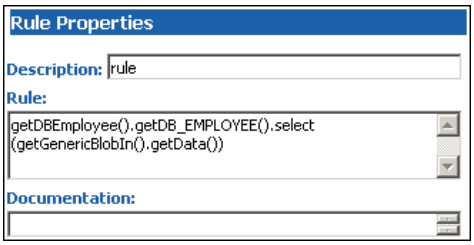
Figure 40 Rule Properties



15 In the **Source Events** pane, expand the **GenericBlobIn** Event Type until the **Data** node is visible.

16 In the **Rule Properties** pane, position the cursor inside the parentheses of the **select** method. Then drag the **Data** node from the **Source Events** pane into the **select** method's parentheses. See Figure 41.

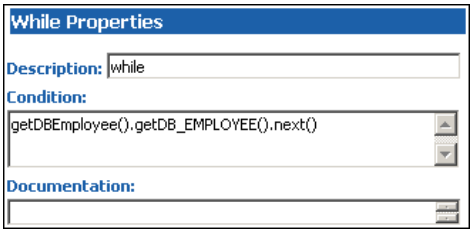
Figure 41 Rule Properties (Continued)



17 Select the newly edited rule in the **Business Rules** pane and click the **while** button to add a new while loop beneath the current rule.

18 Drag the **next** method from the **Destination Events** pane into the **Condition** field of the **While Properties** pane. Click **OK** to close the dialog box. See Figure 42.

Figure 42 While Properties

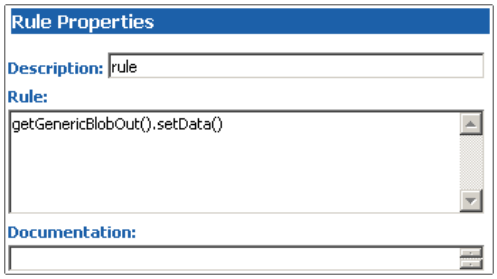


19 Select the newly edited while loop in the Business Rules pane and click the **rule** button to add a new rule as a **child** to the while loop.

20 In the **Destination Events** pane, expand the **GenericBlobOut** Event Type until the **Data** node is visible.

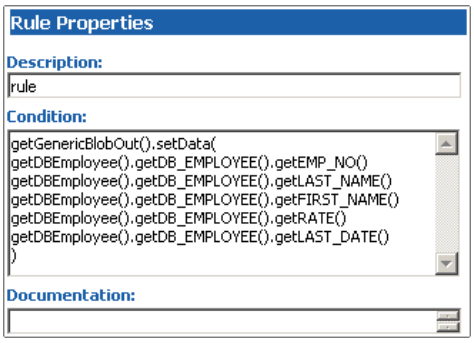
21 Drag the **Data** node into the **Rule** field of the Rule Properties pane. See Figure 43.

Figure 43 Rule Properties



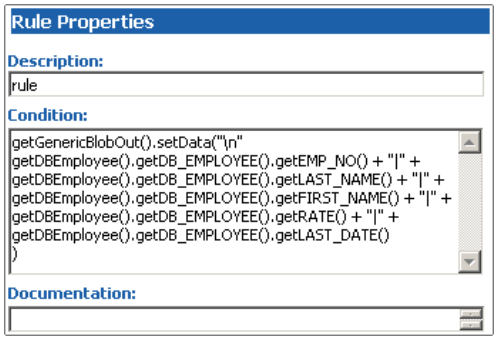
22 In the Rule Properties pane, position the cursor inside the parentheses of the **setData()** method. Then drag each of the five data nodes of **DB_EMPLOYEE** from the Source Events into the parentheses of the rule. See Figure 44.

Figure 44 Rule Properties (Continued)



23 Edit the text of the condition to add a newline character and pipe (|) delimiters between each of the five data nodes. See Figure 45.

Figure 45 Rule Properties (Continued)

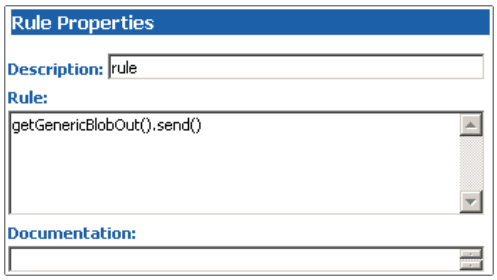


24 Select the newly edited rule in the **Business Rules** pane and click the **rule** button to add a new rule inside the while loop.

25 Drag the root node of the **GenericBlobOut** Event into the **rule** field in the **Rule Properties** pane.

26 Edit the rule; add a **send()** method as shown in Figure 46.

Figure 46 GenericBlobOut iqPut()



27 From the **File** menu, choose **Save** to save the file.

28 From the **File** menu, choose **Compile** to compile the Collaboration.

View the bottom pane to ensure that there were no compiler errors.

29 From the **File** menu, choose **Exit** to close the Java Collaboration Rules Editor and return to the Collaboration Rule.

Note that the **Collaboration Rules** and **Initialization file** fields have been completed by closing the Java Collaboration Editor.

30 Click **OK** to save and close the **DBSelect** Collaboration Rule.

4.4.4 Add and Configure the e*Ways

The sample scenario uses three e*Ways:

- **FileIn** – This e*Way retrieves an Event (text file) containing the database select criteria and publishes it to the **Q1** IQ.
- **DBSelect** – This e*Way retrieves the Generic Event (**Blob**) from the **Q1** IQ. This triggers the e*Way to request information from the external database (via the e*Way Connection) and publishes the results to the **Q2** IQ.
- **FileOut** – This e*Way retrieves the Generic Event (**Blob**) from the **Q2** IQ then writes it out to a text file on the local file system.

To create the FileIn e*Way

1 In the Components pane of the Schema Designer, select the Control Broker click **New e*Way**.

2 Enter **FileIn** for the component name and click **OK**.

3 Select the newly created e*Way and click **Properties** to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 6.

Table 6 FileIn e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	default
Outbound (send) settings	All	default
Poller (inbound) settings	PollDirectory	c:\egate\data\dbSelect_In
	All others	default
Performance Testing	All	default

7 Select **Save** from the **File** menu.

8 Enter **FileIn** as the file name and click **Save**.

9 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the e*Way configuration file editor.

10 On the **Start Up** tab of the e*Way **Properties** window, select **Start automatically**.

11 Click **OK** to save the e*Way properties.

To create the DBSelect e*Way

1 In the Components pane of the Schema Designer, select the Control Broker and click **New e*Way** .

2 Enter **DBselect** for the component name and click **OK**.

3 Select the newly created e*Way and click **Properties** to display the e*Way's properties.

4 Click **Find**.

5 Select **stceway.exe** as the executable file.

6 Click **New** to create a new configuration file.

7 Enter the parameters for the e*Way as shown in Table 7.

Table 7 DBSelect e*Way Parameters

Section Name	Parameter	Value
JVM Settings	All	default

8 Select **Save** from the **File** menu. Enter **DBSelect** as the file name and click **Save**.

9 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the configuration file editor.

10 In the **Start Up** tab of the Business Object Broker properties, select **Start automatically**.

11 Click **OK** to save the e*Way's properties.

To create the FileOut e*Way

1 In the Components pane of the Schema Designer, select the Control Broker and click **New e*Way**.

2 Enter **FileOut** for the component name and click **OK**.

3 Select the newly created e*Way and click **Properties** to display the e*Way's properties.

4 Click **Find**.

5 Select **stcewfile.exe** as the executable file.

6 Click **New** to create a new configuration file.

7 Enter the parameters for the e*Way as shown in Table 8.

Table 8 FileOut e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	NO
	AllowOutgoing	YES
	PerformanceTesting	default

Section Name	Parameter	Value
Outbound (send) settings	OutputDirectory	c:\egate\data\dbSelect_Out
	OutputFileName	dbSelect%d.dat
Poller (inbound) settings	All	default
Performance Testing	All	default

8 Select **Save** from the **File** menu. Enter **FileOut** as the file name and click **Save**.

9 Select **Promote to Run Time** from the **File** menu. Click **OK** to close the configuration file editor.

10 In the **Start Up** tab of the e*Way properties, select **Start automatically**.

11 Click **OK** to save the e*Way properties.

4.4.5 Add and Configure the e*Way Connections

The sample scenario uses one e*Way Connection:

- **JDBCODBC_eWc** – This e*Way Connection connects the **DBselect** component to the external database and returns the requested records to be published to the **Q2** IQ.

To create the e*Way Connection

1 In the Components pane of the Schema Designer, select the **e*Way Connections** folder.

2 Click **New e*Way Connection**.

3 Enter **JDBCODBC_eWc** for the component name and click **OK**.

4 Select the newly created e*Way Connection and click **Properties**.

5 Select **JDBCODBC** from the e*Way Connection Type drop-down list.

6 Enter the timeout parameter of the e*Way by entering a number in milliseconds in the Event Type “get” Interval dialog box. The default value is 10000.

7 Click **New** to create a new configuration file.

8 Enter the parameters for the e*Way Connection.

9 Select **Save** from the **File** menu.

10 Enter **JDBCODBC_eWc** as the file name and click **Save**.

11 Select **Promote to Run Time** from the **File** menu.

12 Click **OK** to save the e*Way Connection’s properties.

4.4.6 Add the IQs

The sample scenario uses two IQs:

- **Q1** – This IQ queues the inbound Events for the DBSelect e*Way.

- **Q2** – This IQ queues the outbound Events for the FileOut e*Way.

To add the IQs

1. In the components pane of the Schema Designer, select the IQ Manager.
2. Click **New IQ** to add a new IQ.
3. Enter the name **Q1** and click **Apply** to save the IQ and leave the New IQ dialog box open.
4. Enter the name **Q2** and click **OK** to save the second IQ.
5. Select the IQ Manger and click **Properties**.
6. Select the **Start automatically** check box and click **OK** to save the properties.

4.4.7 Add and Configure the Collaborations

The sample scenario uses three Collaborations:

- **FileIn_PassThru** – This Collaboration uses the **GenericPassThru** Collaboration Rule.
- **DBselect_collab** – This Collaboration uses the **GenericEventToDatabase** Collaboration Rule to execute the **dbCollab.class** Java Collaboration file.
- **FileOut_PassThru** – This Collaboration uses the **GenericPassThru** Collaboration Rule.

To add the FileIn_PassThru Collaboration

1. In the components pane of the Schema Designer, select the **FileIn** e*Way.
2. Click **New Collaboration** to create a new Collaboration.
3. Enter the name **FileIn_PassThru** and click **OK**.
4. Select the newly created Collaboration and click **Properties** .
5. Select **GenericPassThru** from the drop-down list of Collaboration Rules.
6. Click the upper **Add** button to add a new Subscription.
7. Select the **GenericEvent** Event Type and the **<External>** source.
8. Click the lower **Add** button to add a new Publication
9. Select the **GenericEvent** Event Type and the **Q1** destination.
10. Click **OK** to close the Collaboration's properties.

To add the DBselect_collab Collaboration

1. In the components pane of the Schema Designer, select the **DBSelect** e*Way.
2. Click the **New Collaboration** button to create a new Collaboration.
3. Enter the name **DBselect_collab** and click **OK**.
4. Select the newly created Collaboration and click **Properties**.
5. Select **DBSelect** from the drop-down list of Collaboration Rules.

6 Click **Add** (upper part of the window) to add a new Subscription.

7 Select the **GenericEvent** Event Type and the **FileIn_PassThru** source.

8 Click **Add** (lower part of the window) to add a new Publication.

9 Select the **DBEmployee** Event Type and the **JDBCODBC_eWc** destination.

10 Click **Add** (lower part of the screen) to add a new Publication.

11 Select the **GenericEvent** Event Type and the **Q2** destination.

12 Click **OK** to close the Collaboration's properties.

To add the FileOut_PassThru Collaboration

1 In the components pane of the Schema Designer, select the **FileOut** e*Way.

2 Click **New Collaboration**.

3 Enter the name **FileOut_PassThru** and click **OK**.

4 Select the newly created Collaboration and click **Properties**.

5 Select **GenericPassThru** from the drop-down list of Collaboration Rules.

6 Click **Add** (upper part of the window) to add a new Subscription.

7 Select the **GenericEvent** Event Type and the **DBSelect_collab** source.

8 Click **Add** (lower part of the window) to add a new Publication.

9 Select the **GenericEvent** Event Type and the **<External>** destination.

10 Click **OK** to close the Collaboration's properties.

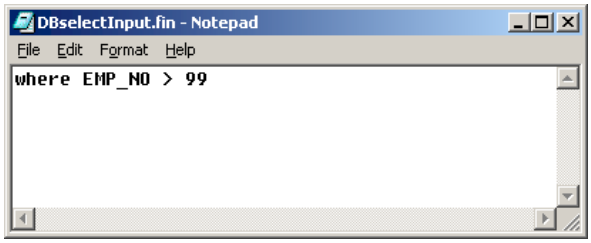
4.4.8 Run the Schema

You must create a sample imput file to run the schema. Once you have created the input file, you can start the Control Broker from a command prompt to execute the Schema. After running the Schema, view the output text file to verify the results.

The sample input file

Use a text editor to create an input file that the inbound file e*Way (**FileIn**) will read. This simple input file contains the criteria for the **dbSelect.class** Collaboration's select statement. An example of an input file is shown in Figure 47.

Figure 47 Sample Input File



To start the Control Broker

From a command prompt, type the following command:

```
stccb -ln logical_name -rh registry -rs DBSelect -un user_name
-up password
```

where

logical_name is the logical name of the Control Broker,

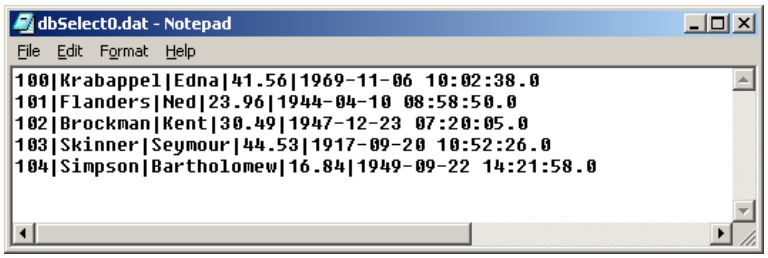
registry is the name of the Registry Host, and

user_name and ***password*** are a valid e*Gate username/password combination.

To verify the results

Use a text editor to view the output file **c:\eGate\data\dbSelect_out\dbSelect0.dat**. Figure 48 shows an example of the records that were returned by the sample schema.

Figure 48 Sample Output File



4.5 The Empty ETD

If you are unable to connect to the Database Wizard using a valid ODBC driver, we have provided an empty ETD for you so that you can build your collaboration using the Collaboration Editor.

The empty ETD, **EmptyETD.xsc**, is located at **<egate>/server/registry/repository/default/etd/db/**

To use the empty ETD, you need to use SqlStatementAgent:

1. Declare SqlStatementAgent with a specific name using the method getSqlStatementAgent or the method getPreparedSqlAgent.
2. Specify and execute a SQL statement.

The following sample briefly illustrates how to build a collaboration using the empty ETD. In the sample collaboration, we copy new rows from the table DB_EMPLOYEE to the table EMPLOYEE_UPDATE whenever new records in the table EMPLOYEE are found.

The table description of DB_EMPLOYEE is shown in Table 9.

Table 9 DB_EMPLOYEE

Name	Null	Type
EMP_NO		NUMBER(38)
LAST_NAME		VARCHAR2(30)
FIRST_NAME		VARCHAR2(30)
RATE		FLOAT(126)
LAST_UPDATE		DATE

The table description of EMPLOYEE_UPDATE is shown in Table 10.

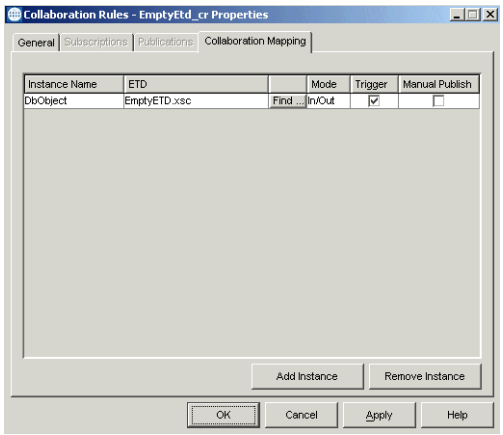
Table 10 EMPLOYEE_UPDATE

Name	Null	Type
EMP_NO		NUMBER(38)
LAST_NAME		VARCHAR2(30)
RATE		FLOAT(126)

To build a Collaboration using the Collaboration Editor

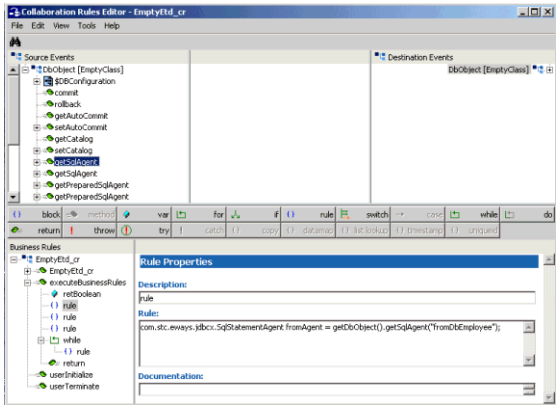
1. Create a new Java Collaboration.
2. Select the provided empty ETD in "Collaboration Mapping" as in Figure 49.

Figure 49 EmptyETD_crProperties



3. Click on the tab General in the above Collaboration Mapping window, click the New button to open the collaboration editor window.
4. Drag and drop the method getSQLAgent into the first collaboration rule. You can drag and drop a getPreparedSqlAgent method if you have a prepared SQL statement. Label it with a name "fromDbEmployee". This will create a SqlStatementAgent object, which we assigned to variable "fromAgent". See Figure 50.

Figure 50 Collaboration Rules Editor - EmptyETD_cr

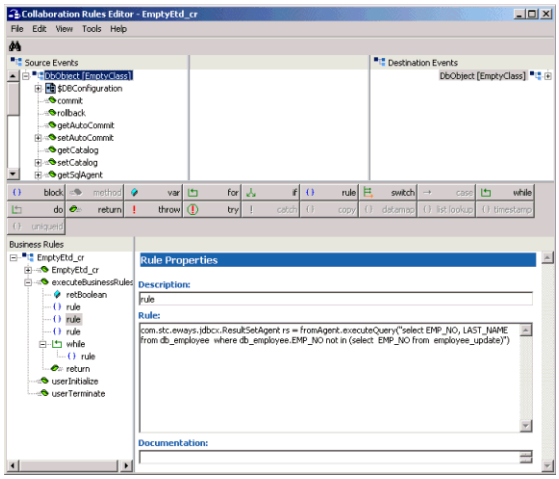


5 In the second collaboration rule, we use the above SqlStatementAgent object to execute this SQL statement:

```
select EMP_NO, LAST_NAME from db_employee where db_employee.EMP_NO
not in(select EMP_NO from employee_update)
```

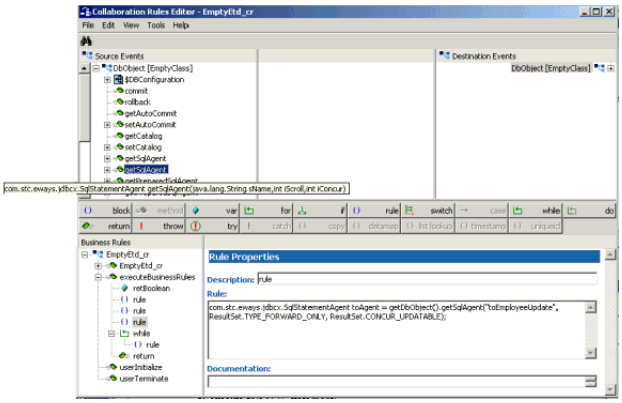
which selects new records from the DB_EMPLOYEE table that don't exist in the EMPLOYEE_UPDATE table. The execution of the query will return a ResultSetAgent object, which we assigned to the variable "rs".

Figure 51 Collaboration Rules Editor - EmptyETD_cr, Step 5



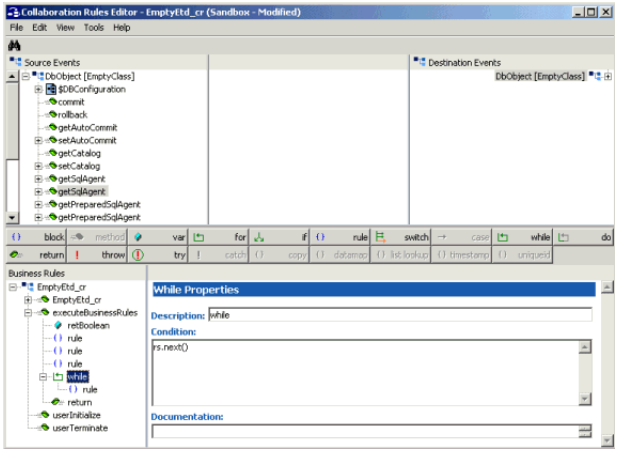
6 Drag and drop the method getSqlAgent which has three parameters into the collaboration rule. For the third parameter we pass in the value "ResultSet.CONCUR_UPDATABLE" so that later we can use this SqlStatementAgent to update database. Label it with a name "toEmployeeUpdate". This will create a SqlStatementAgent object, which we assigned to variable "toAgent". See Figure 52.

Figure 52 Collaboration Rules Editor - EmptyETD_cr, Step 6



7 Next, we create a while loop to loop through the previously created ResultSetAgent object “rs”.

Figure 53 Collaboration Rules Editor - EmptyETD_cr, Step 7

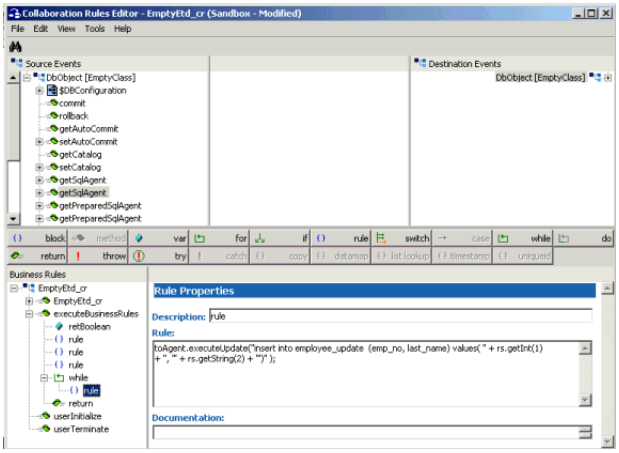


8 Inside the loop, we use the above SqlStatementAgent object to execute this runtime created SQL statement:

```
"insert into employee_update  (emp_no, last_name) values( " +
rs.getInt(1)
+ ", '" + rs.getString(2) + "')"
```

which gets a record from the ResultSetAgent rs and inserts it into the table EMPLOYEE_UPDATE

Figure 54 Collaboration Rules Editor - EmptyETD_cr, Step 9



The sample schema for the above has been included in the samples directory on the e*Gate installation disk \samples\ewjdbcodbc\EmptyDBETD.zip. To import the sample schema, do the following:

1 From the Enterprise Manger toolbar, click File.

2 Click Import Definitions from File...

3 From the Import Definitions window, click Next.

4 From the Import Wizard - Step 1 window, select Schema and click Next.

5 From the Import Wizard - Step 2 window, click Select to navigate to the Samples folder located on the Installation CD-ROM.

6 Highlight the file EmptyDBETD.zip and click Next.

If you use **getSQLAgent** and **getPreparedSQLAgent**, with multiple connections within the same ETD, for example connect, disconnect, then connect again, the name you used for the SQLAgent should only be used in the scope of one connection period. Even if the you pass the same SQL statement string.

4.6 Troubleshooting the JDBC/ODBC Java e*Way

This section should provide you with additional support when troubleshooting your e*Way.

The performance and functionality of the JDBC/ODBC e*Way will depend on the driver(s) selected. Certain drivers may not support all JDBC features. Consult the documentation for your respective drivers) for more information.

1 When using DataDirect JDBC driver with some databases, you will need to specify "cursor" for the connection attribute "SelectMethod" for the following reasons:

A This e*Way implementation uses the same JDBC connection to create multiple JDBC statements.

B and the default transaction behavior is set to "false" for Autocommit.

DataDirect's JDBC driver cannot use the same JDBC connection to create more than one statement when Autocommit is set to false. For additional information regarding this, please see your DataDirect JDBC driver documentation.

2 Using fully qualified table names can throw a SQL syntax error or exception while using Informix. Try using short names.

3 When using a combination of database native ODBC drivers and the Sun JDBC - ODBC Bridge drivers, unpredictable results may occur. Try using a driver type that does not require database native libraries. Type 1 and 2 drivers have limited functionality and are unable to take full advantage of Java feature sets.

4 Updatable ResultSets are not supported by Sun's JDBC-ODBC Bridge driver. This is a Type 1 driver. Type 1 and 2 drivers have limited functionality and are unable to take full advantage of Java feature sets. Try using a Type 3 or 4 driver.

5 When using non-XA compliant drivers in XA mode throws a java.sql.SQL Exception. Try using an XA compliant driver to avoid this problem.

6 If you are using an AIX 4.3.3 Korean supported operating system, the Merant JDBC/ODBC bridge driver is expecting member libodbc.o in libodbc.a. Check the member name in your libodbc.a by doing the following:

```
dump -H libodbc.a
```

If your libodbc.a has a different member name for libodbc.o, for example, odbc.so, then perform the following operations after saving your original libodbc.a in a backup directory.

To extract and rename the member, do the following:

```
ar -p libodbc.a odbc.so > libodbc.o
```

To add the libodbc.o member to libodbc.a, then do the following:

```
ar -v -r libodbc.a libodbc.o
```

Failure to do this will result in an error message as follows: 'No Suitable Driver' resulting in the e*Way not connecting to the database.

7 For Korean Solaris 8 operating systems, the JDBC/ODBC bridge is expecting different names for the shared libraries. Thus, the user must create a softlink as follows in the Merant ODBC drivers directory:

```
ln -s libodbcinst.so libodbcinst.so.1
 ln -s libodbc.so libodbc.so.1
```

Failure to do this will result in the following error message: 'No Suitable Driver' and the e*Way will be unable to connect to the database.

8 Some type 4 drivers support Double Byte Character Set (DBCS) table names and column names within the tables. If the current driver you are using does not support DBCS, you will see unexpected results.

4.7 Monk ODBC Implementation

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

This Section Explains:

4.8 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 a 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 a SQL Stored Procedure. This option is only available for *Delimited* messages.

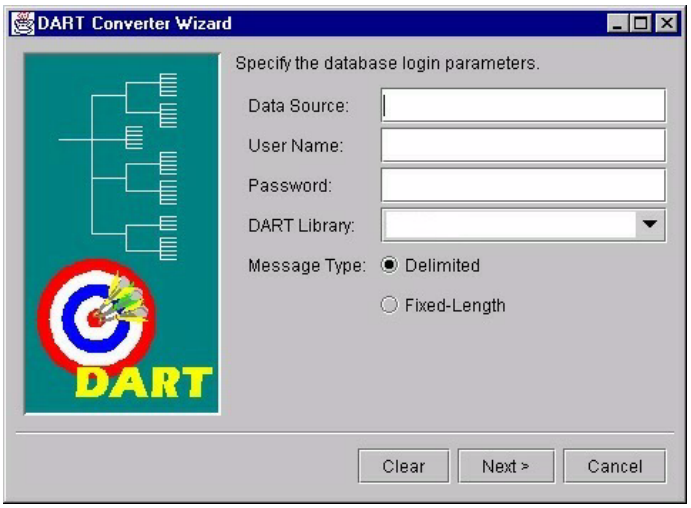
The results of these three types of message criteria are explained in **"The Event Type Definition Files" on page 112**.

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 **Finish**.
7. The Converter Wizard will launch.

Figure 55 Converter Wizard Subordinate Dialog Box



8 Enter the Data Source.

9 Enter the User Name.

10 Enter the Password.

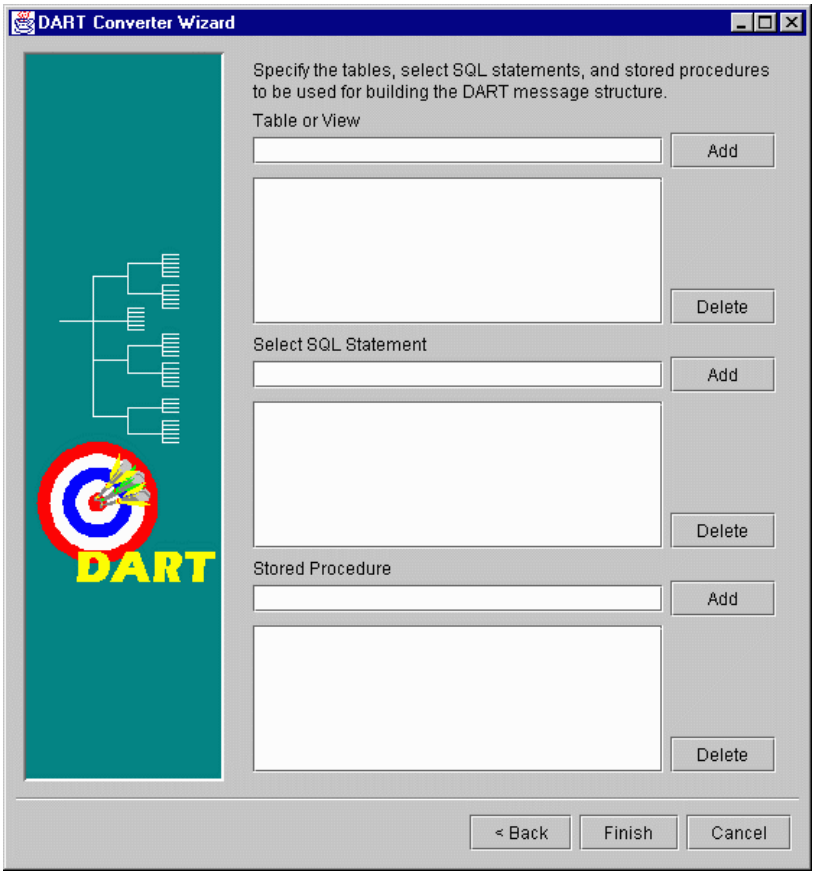
11 Select the DART Library. You must have the corresponding e*Way installed prior to your selection.

12 Select the correct Message Type.

Note: *It is important to enter the correct Data Source and Message Type. For Oracle the Data Source is in the Servicename.world format*
The Fixed-length Message Type is used for DART bulk insert only.
The Delimited Message Type is for all other DART structure calls.
See **Figure 55 on page 110**

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

Figure 56 Converter Wizard Delimited Message Type Dialog Box



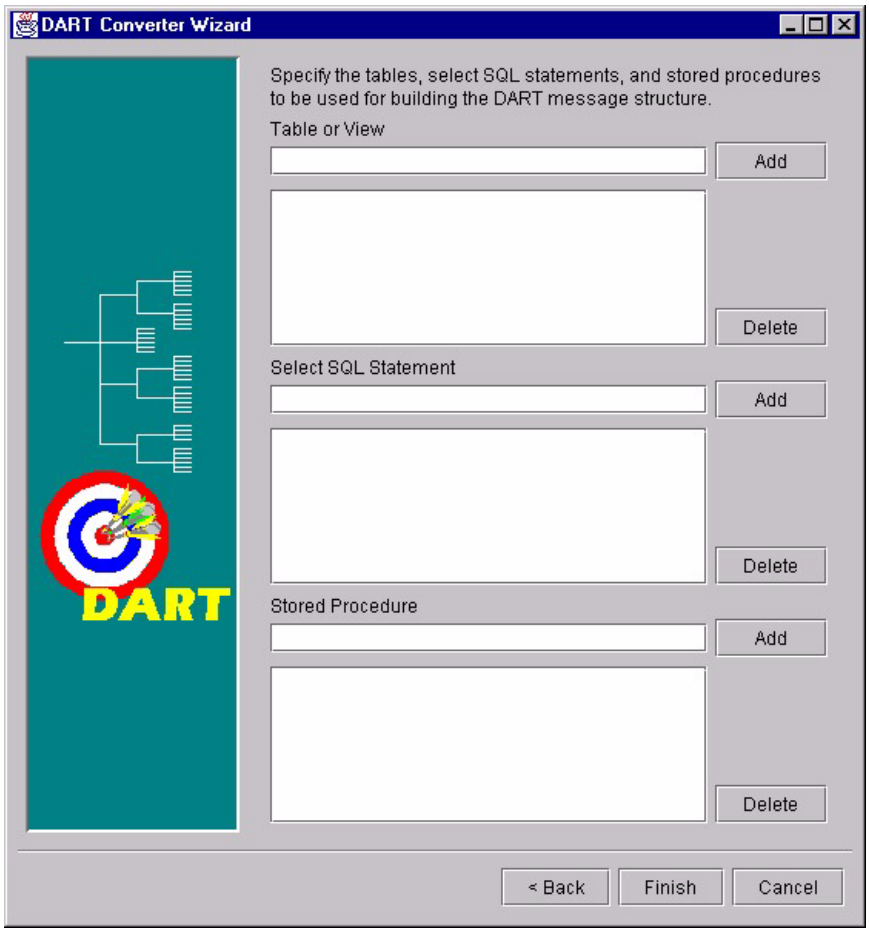
13 Select or Add the correct Table or View

14 Select or Add the correct SQL Statement

15 Select or Add the correct Stored Procedure.

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

Figure 57 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 Oracle 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.*

For Oracle, ($), or (#) can be used in a name, although the Oracle User's Guide strongly discourages their use.

4.8.1 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 56 on page 111** and **Figure 57 on page 112**).

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 58 generates the ETD shown in Figure 59.

Figure 58 Table or View Selection

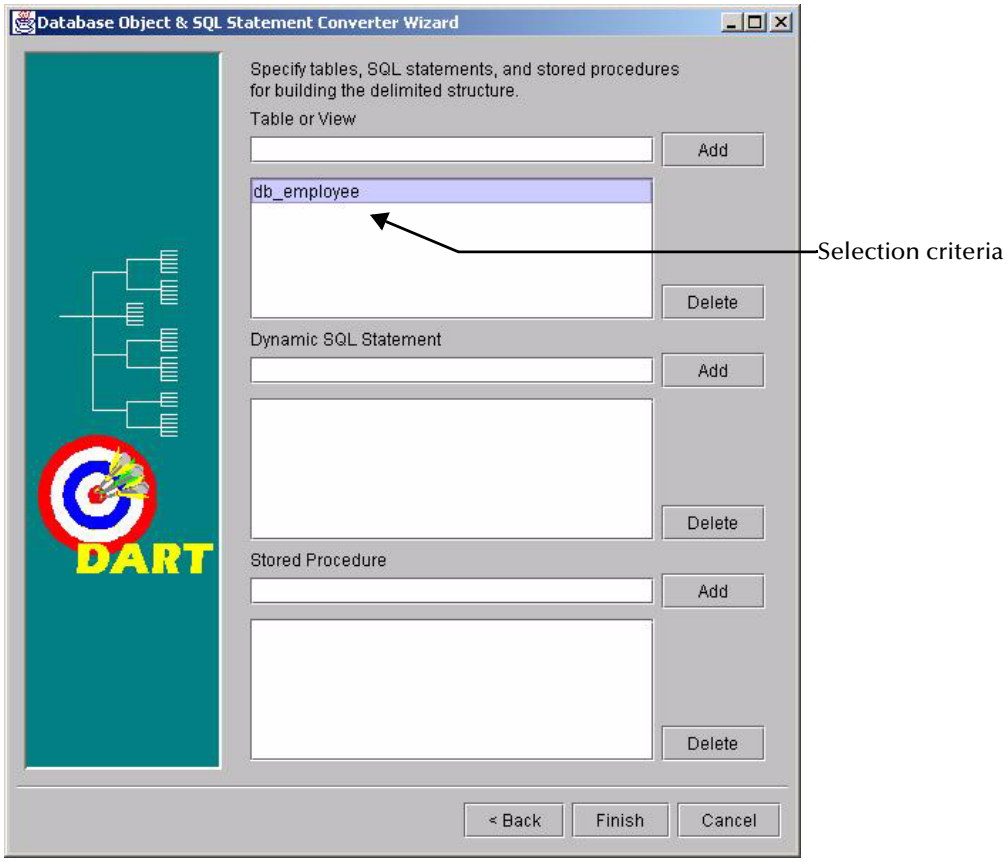
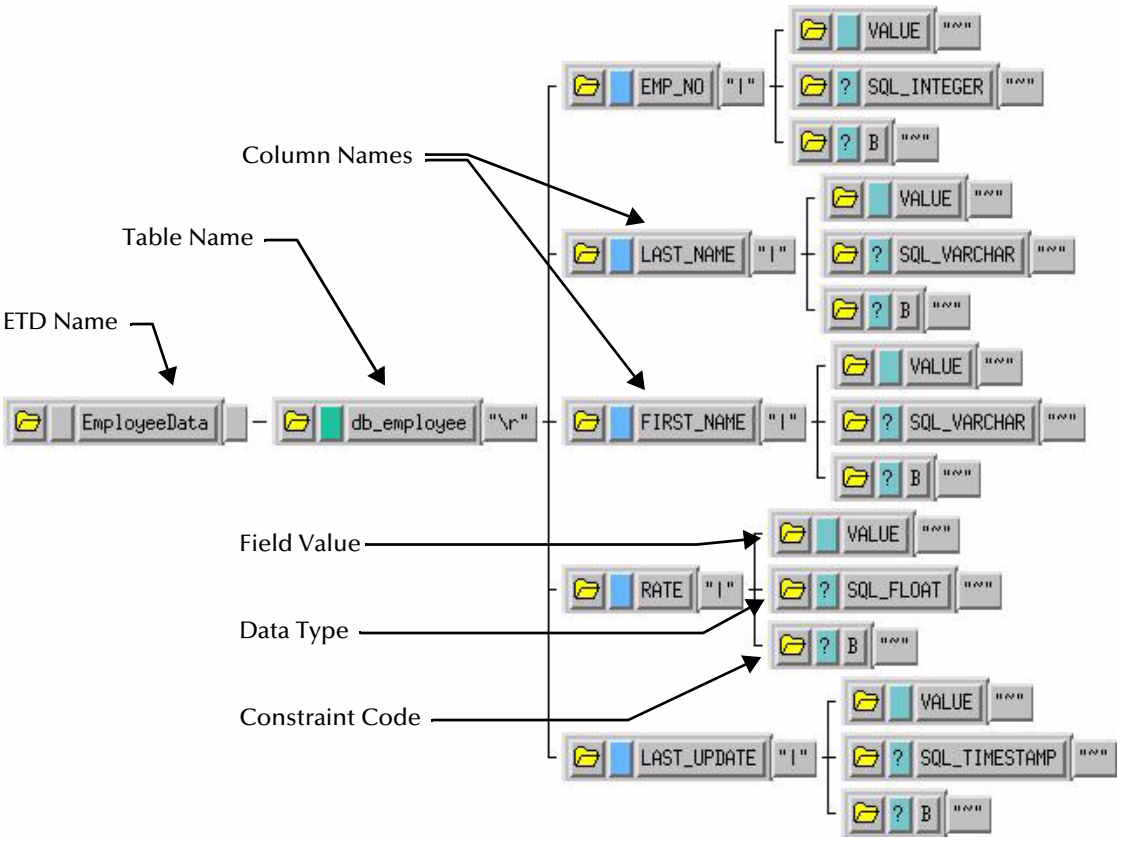


Figure 59 Table or View ETD



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 11 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 405 and **db-struct-execute** on page 447.

The SQL statement shown in Figure 60 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 61 on page 116**.

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

Figure 60 Dynamic SQL Statement Select

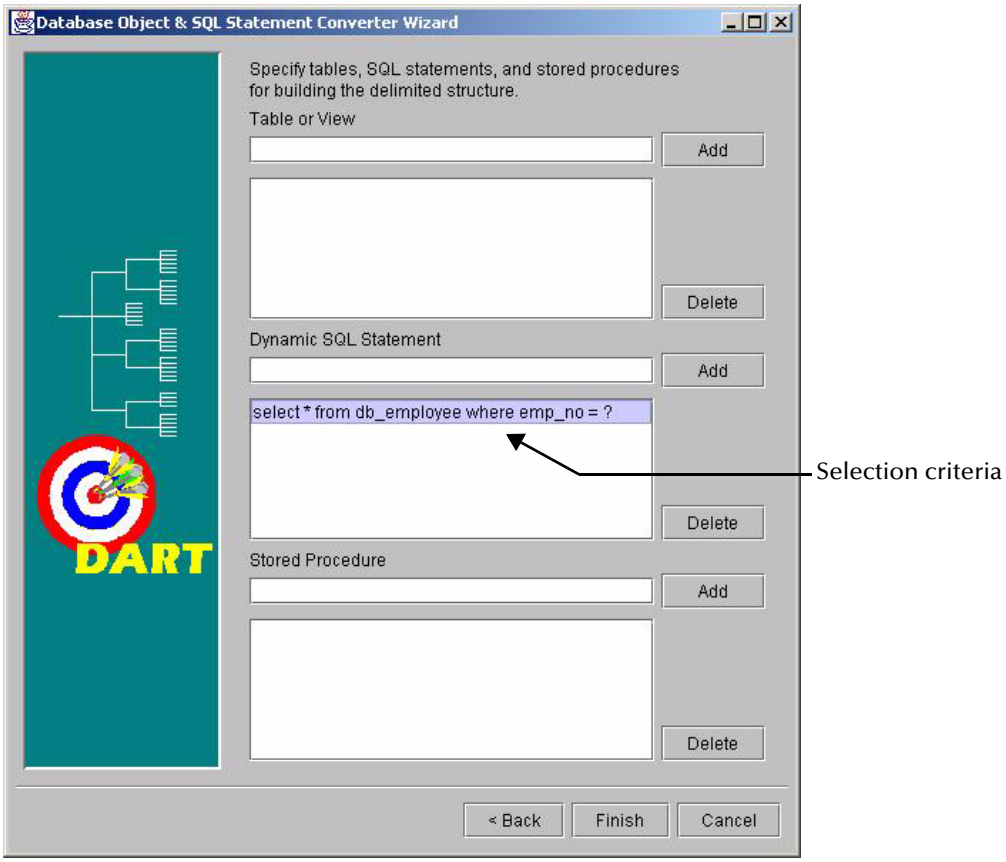
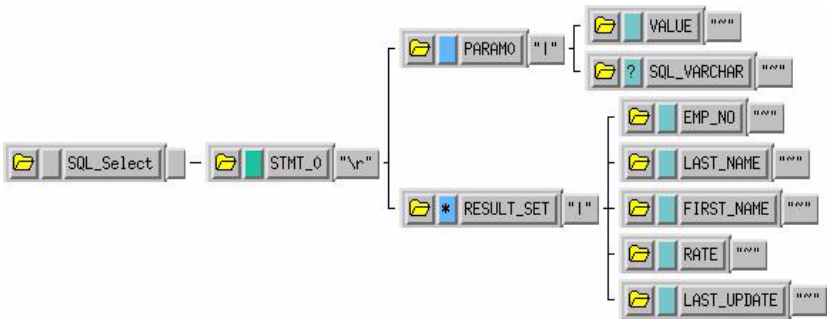


Figure 61 Dynamic SQL Statement ETD



The **PARAM0** node in the ETD shown in Figure 61 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 62 generates an the ETD shown in Figure 63. Below is the contents of the sample stored procedure:

```
procedure VARIABLE_NUM_NEW_PROC
(
    BATCH_SIZE          in integer,
    FOUND               in out integer,
    DONE_FETCH          out integer,
    INT_RET             out integer,
    FLOAT_RET           out float,
    SMALL_INT_RET       out smallint,
    DOUBLE_RET          out double precision,
    REAL_RET            out real,
    DECIMAL_RET         out decimal,
    DECIMAL_PRECISE_RET out decimal,
    NUMBER_RET          out number,
    NUMBER_PRECISE_RET  out number
)
as
  temp int := 0;
  cursor GET_COUNT IS
    select count(*) from NUM_TABLE;
  cursor GET_TYPE IS
   select INT_NUM, FLOAT_NUM, SMALL_INT_NUM, DOUBLE_NUM, REAL_NUM,
        DECIMAL_NUM, DECIMAL_PRECISE, NUM_NUM, NUM_PRECISE
     from NUMBER_TYPE;
begin
  OPEN GET_COUNT;
    fetch GET_COUNT into temp;
  CLOSE GET_COUNT;
```

```
  DONE_FETCH := temp;
  if temp = 0
  then
    FOUND := 0;
  else
    FOUND := 1;
  end if;
  OPEN GET_TYPE;
    fetch GET_TYPE into INT_RET, FLOAT_RET, SMALL_INT_RET, DOUBLE_RET,
        REAL_RET, DECIMAL_RET, DECIMAL_PRECISE_RET, NUMBER_RET,
        NUMBER_PRECISE_RET;
  CLOSE GET_TYPE;
end;
```

Figure 62 Stored Procedure Selection

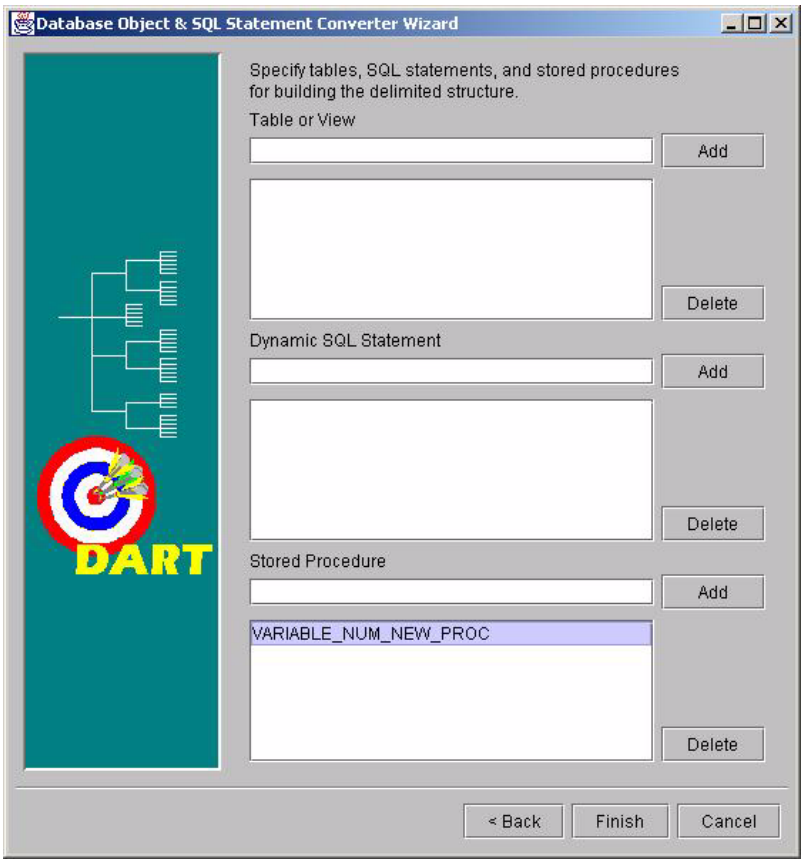
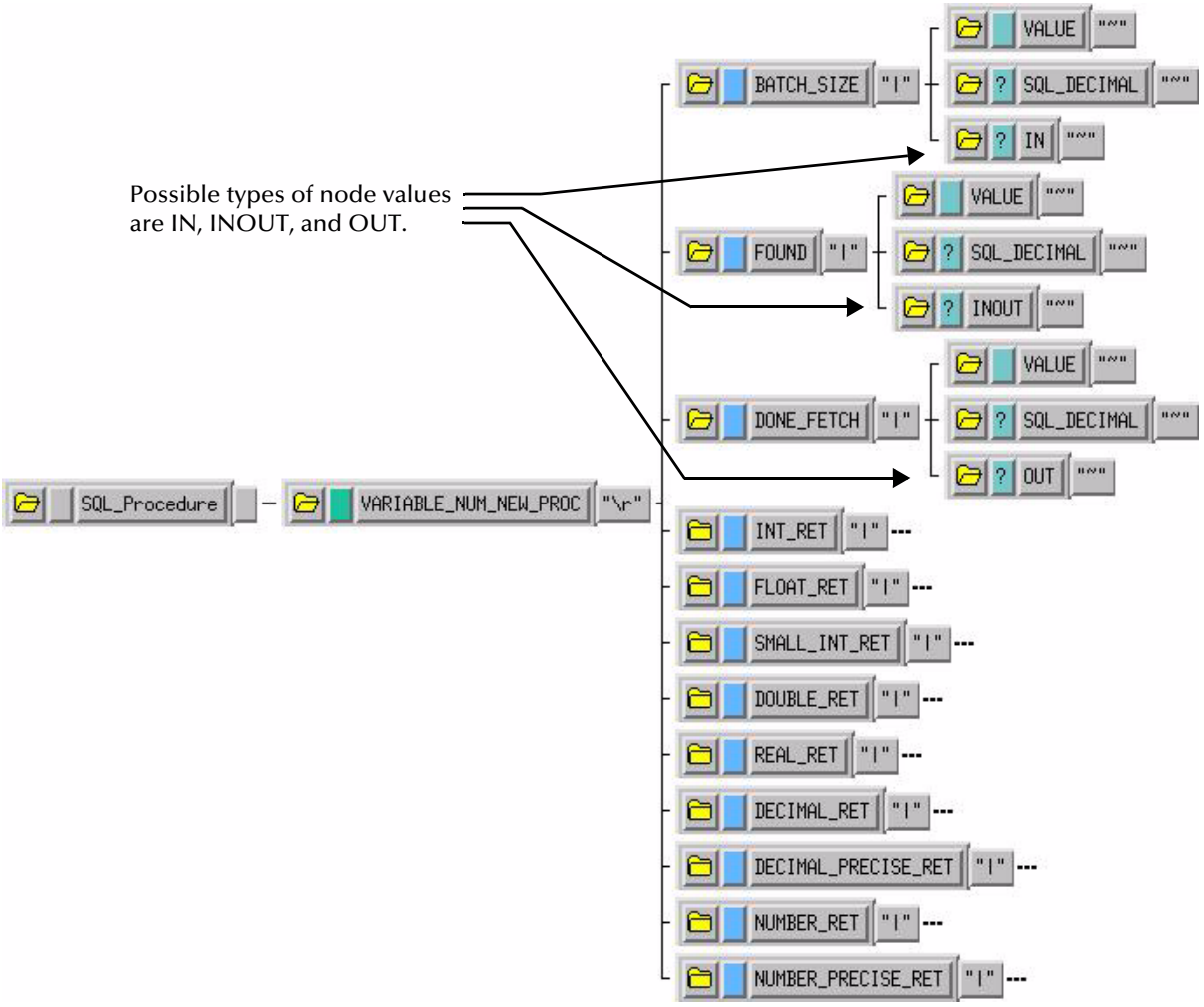


Figure 63 Stored Procedure ETD



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 results are returned to the "OUT" or "INOUT" nodes.

4.9 Vendor-Specific Driver Notes

Certain functions are known to behave differently based on the type of ODBC drivers being used on the client machine. These differences in functionality are explained below.

4.9.1 IBM ODBC DB2 Drivers

The following issues are known to exist with the IBM DB2 ODBC drivers.

Support for BLOB and CLOB Data Types

The IBM ODBC DB2 drivers support BLOB (Binary Large Object) or CLOB (Character Large Object) data only if specifically configured to do so.

Note: *Large records (CLOB or BLOB) should be inserted by using the* ***db-stmt-bind*** *and* ***db-stmt-execute*** *functions. For more information on these functions, see* **"db-stmt-bind" on page 405** *and* **"db-stmt-execute" on page 410***.*

Follow the procedures below for configuring your client workstation to support BLOB and CLOB data.

Configuring a Windows NT / 2000 Client to Support Long Objects

1. Open the **Control Panel**.
2. Open the **Administrative Tools** (Windows 2000 only).
3. Open the **ODBC Data Sources**.
4. Select the **System DSN** tab.
5. Select appropriate driver for your IBM DB2 implementation (such as "IBM_UDB2") and click **Configure**.
6. Enter the appropriate username and password to connect to the DB2 data source.
7. Select the **Data Type** tab.
8. Select the **Long object binary treatment** parameter.
9. Choose the **As LONGVAR data** setting.
10. Save the settings and close the ODBC configuration utility.

Configuring a UNIX Client to Support Long Objects

1. Use a text editor to open the **db2cli.ini** file.
2. Add the following line:

```
LONGDATACOMPAT = 1
```

Note: *Large records using the CLOB data type are limited to approximately 1GB in size. Larger records should not be inserted.*

4.9.2 Merant ODBC Drivers

The following issues are known to exist with the Merant ODBC drivers.

Support for BLOB and CLOB Data Types

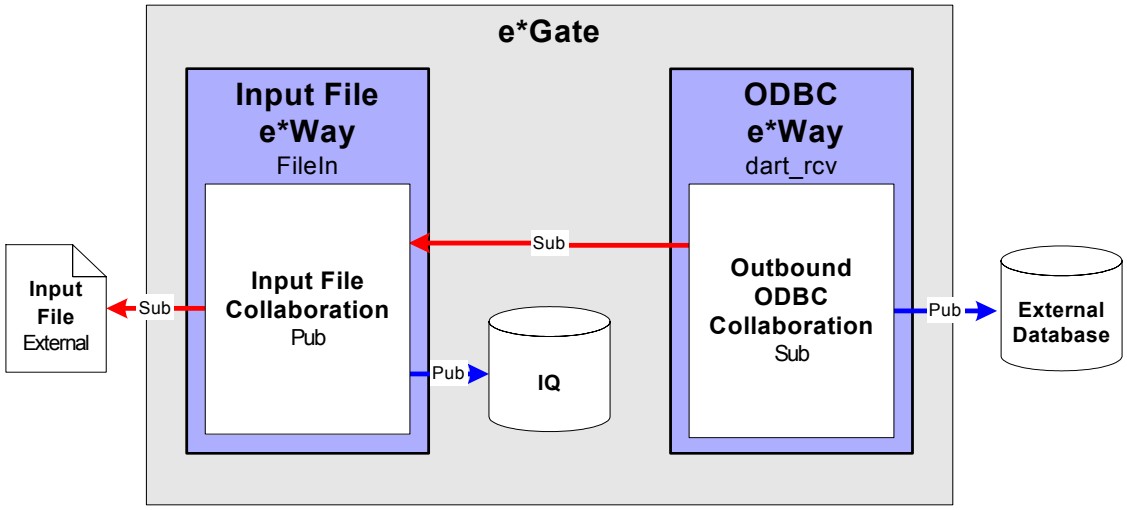
The Merant ODBC drivers do not support BLOB (Binary Large Object) or CLOB (Character Large Object) data. This affects the size and type of large records that can be inserted into tables from a client using the Merant ODBC drivers.

4.10 Sample One—Publishing e*Gate Events to an ODBC Database

This section describes how to use the ODBC e*Way in a sample implementation. This sample schema demonstrates the publishing of e*Gate Events to an external database.

This scenario uses a file e*Way to load an input file containing employee information and generate the initial Event. The ODBC e*Way subscribes to the Event and inserts the employee records into the external database.

Figure 64 Publishing to an External Database

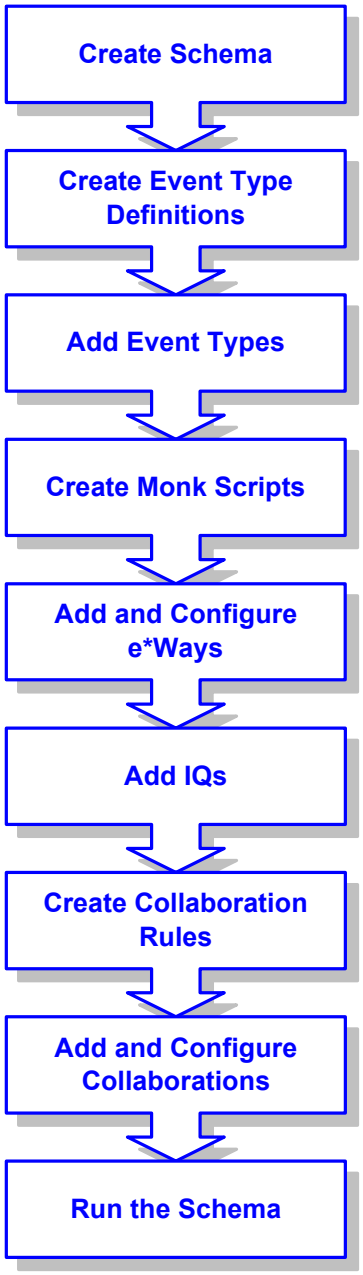


Overview of Steps

The sample implementation follows these general steps:

- "Create the Schema" on page 121
- "Create the Event Type Definitions" on page 122
- "Add the Event Types" on page 123
- "Create the Monk Scripts" on page 124
- "Add and Configure the e*Ways" on page 124
- "Add the IQs" on page 126
- "Create the Collaboration Rules" on page 127
- "Add and Configure the Collaborations" on page 128
- "Run the Schema" on page 129

Figure 65 Schema Configuration Steps



4.10.1 Create the Schema

The first step in deploying the sample implementation is to create a new Schema. After installing the ODBC e*Way Intelligent Adapter, do the following:

1. Launch the e*Gate Schema Designer GUI.
2. When the Schema Designer prompts you to log in, select the Registry Host, User Name, and Password to be used to log in and click **Log In**.
3. From the list of Schemas, click **New** to create a new Schema.

4 For this sample implementation, enter the name **ODBC_Sample1** and click **Open**.

The Schema Designer will launch and display the newly created Schema.

4.10.2 Create the Event Type Definitions

Three Event Type Definitions are used in this sample. The ETDs are:

- **EventMsg.ssc** – This standard ETD is used by the **FileInEvent** Event Type.
- **db_rcv_in.ssc** – This user-created ETD contains basic employee information such as name, rate, and date.
- **db_rcv_struct.ssc** – This user-created ETD contains the same basic employee information formatted appropriately for the external data source.

To create the db_rcv_in ETD:

1 From the e*Gate Schema Designer, click  to launch the ETD Editor.

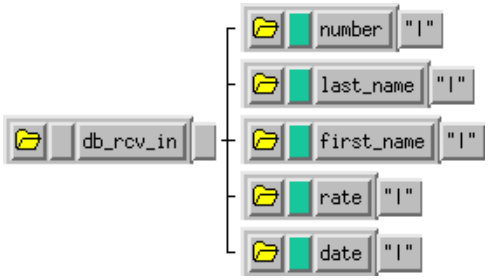
2 Click  to create the new ETD.

The New ETD dialog will be displayed.

3 Enter **db_rcv_in.ssc** as the file name for the ETD.

4 Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 66 The **db_rcv_in.ssc** ETD



5 Click  to save the ETD.

6 From the **File** menu, select **Promote to Run Time**. Click **Yes** to confirm the promotion of the file.

To create the db_rcv_struct ETD:

1 From the e*Gate Schema Designer, click  to launch the ETD Editor.

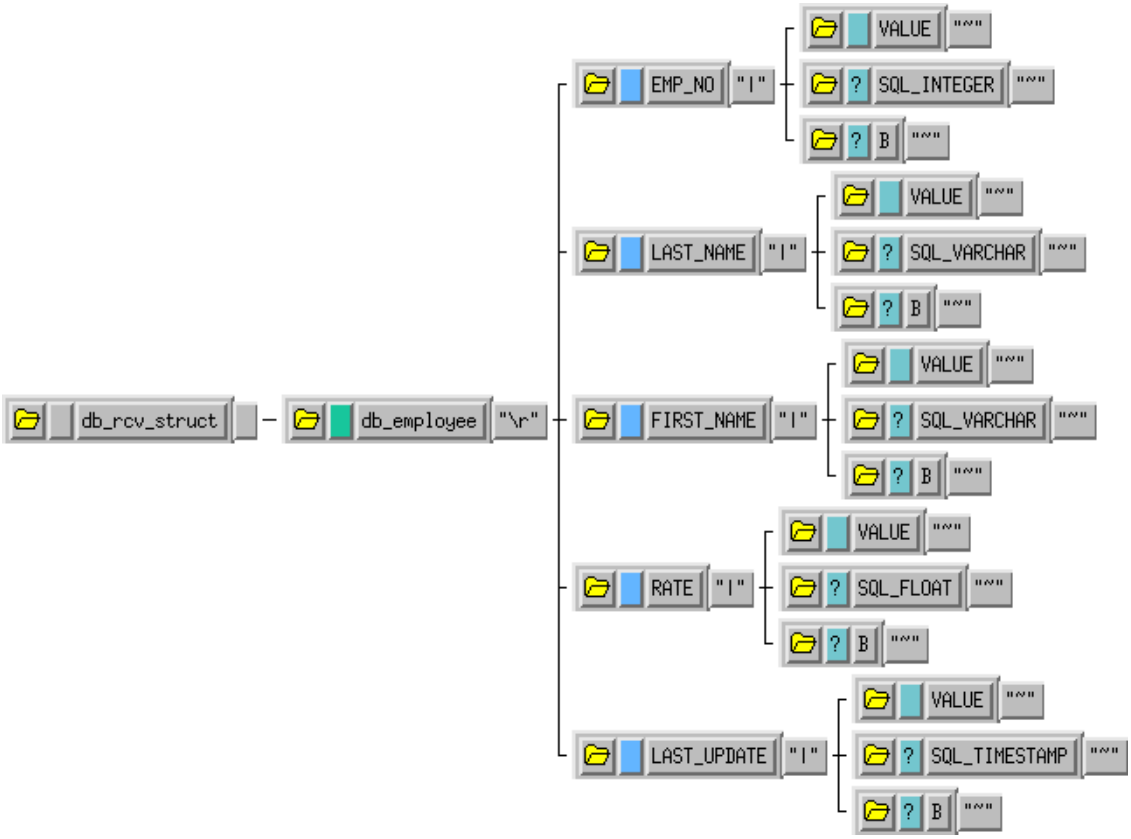
2 Click  to create the new ETD.

The New ETD dialog will be displayed.

3 Enter **db_rcv_struct.ssc** as the file name for the ETD.

4 Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 67 The **db_rcv_struct.ssc** ETD



5 Click  to save the ETD.

6 From the **File** menu, select **Promote to Run Time**. Click **Yes** to confirm the promotion of the file.

4.10.3 Add the Event Types

Three Event Types are used in this sample. The Event Types are:

- **FileInEvent** – This Event Type represents the inbound data from an external input file. This Event Type uses the **EventMsg.ssc** ETD.
- **db_rcv_in** – This Event Type represents the data transported by the input file e*Way. This Event Type uses the **db_rcv_in.ssc** ETD.
- **db_rcv_struct** – This Event Type represents the transformed Event that will be written to the external database. This Event Type uses the **db_rcv_struct.ssc** ETD.

To add the Event Types:

1 In the components pane of the Schema Designer, select the Event Types folder.

2 Click  to add a new Event Type.

3 Enter **FileInEvent** and click **OK**.

4 Select the newly created Event Type and click  to display the Event Type's properties.

5 Click **Find** to display the list of Event Types.

6 Navigate to the **monk_scripts\common** folder, select **EventMsg.ssc**, and click **Select**.

7 Click **OK** to close the Event Type's properties.

Repeat these steps for the **db_rcv_in** and **db_rcv_struct** Event Types using the appropriate Event Type Definition files.

4.10.4 Create the Monk Scripts

This sample implementation uses a DART script (**db_rcv.dsc**) to communicate with the external Oracle database.

To create the DART script:

1 From the e*Gate Schema Designer, click  to launch the Collaboration Rules Editor.

2 Click  to create a new DART script.

The New Collaboration Rules Script dialog will be displayed.

3 Enter the name **db_rcv** (with no file extension) as the **File name**.

4 Select **DART Send** from the list of file types. The extension **.dsc** will be appended to the file name.

5 Click  to display the list of source files. Select **db_rcv_in.ssc** as the source file.

6 Click  to display the list of destination files. Select **db_rcv_struct.ssc** as the destination file.

7 Enter the rules.

8 Click  to save the script.

9 Close the Collaboration Rules Script Editor.

4.10.5 Add and Configure the e*Ways

The sample Schema uses two e*Ways: **FileIn** and **ODBC_rcv**. The **FileIn** e*Way reads in the input data file and queues it for the ODBC e*Way. The **ODBC_rcv** e*Way writes the records to the **db_employee** table in the external database.

To add and configure the FileIn e*Way:

1 In the components pane of the Schema Designer, select the Control Broker and click  to add a new e*Way.

2 Enter **FileIn** for the component name and click **OK**.

3 Select the newly created e*Way and click  to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 12.

Table 12 FileIn e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	YES
	AllowOutgoing	NO
	PerformanceTesting	default
Outbound (send) settings	All	default
Poller (inbound) settings	PollDirectory	c:\egate\data\dart
	InputFileMask	*.dat
	All others	default
Performance Testing	All	default

7 Select **Save** from the **File** menu. Enter **FileIn** as the file name and click **Save**.

8 Select **Promote to Run Time** from the **File** menu. Click **OK** to continue.

9 A message will notify you that the file has been promoted to run time. Click **OK** to close the e*Way configuration file editor.

10 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

11 Click **OK** to save the e*Way properties.

To add and configure the ODBC_rcv e*Way:

1 In the components pane of the Schema Designer, select the Control Broker and click  to add a new e*Way.

2 Enter **ODBC_rcv** for the component name and click **OK**.

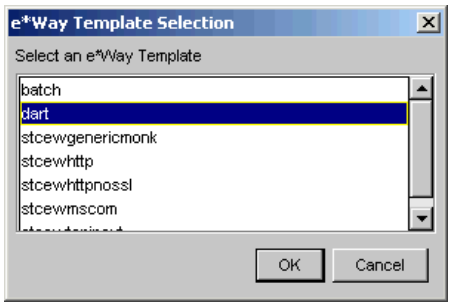
3 Select the newly created e*Way and click  to display the e*Way's properties.

4 Use the **Find** button to select **stcewgenericmonk.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Select the **dart** e*Way template and click OK. See Figure 68.

Figure 68 DART e*Way Template Selection



7 Enter the parameters for the e*Way as shown in Table 13.

Table 13 ODBC_rcv e*Way Parameters

Section Name	Parameter	Value
General Settings	All	default
Communication Setup	Start Exchange Data Schedule	Repeatedly, every 1 minute
	All others	default
Monk Configuration	Process Outgoing Message Function	monk_scripts\common\db_rcv.dsc
	Exchange Data With External Function	monk_scripts\common\db_rcv.dsc
	All others	default
Database Setup	Database Type	ODBC
	All others	Use local settings

Note: *Use the appropriate* ***Database Name****,* ***User Name****, and* ***Encrypted Password*** *according to your local ODBC configuration.*

8 Save the e*Way's configuration file and promote it to run time.

9 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

10 Click **OK** to save the e*Way properties.

4.10.6 Add the IQs

The sample Schema requires one Intelligent Queue—**ODBC_IQ**.

To add the IQ:

1 In the components pane of the Schema Designer, select the IQ manager. Click  to create the new IQ.

2 Enter the name **ODBC_IQ** and click **OK** to save the IQ.

3 Select the IQ Manager and click  to display the IQ Manager's properties.

4 In the **Start Up** tab of the IQ Manager's properties, select the **Start automatically** check box.

5 Click **OK** to save the IQ Manager's properties.

4.10.7 Create the Collaboration Rules

This sample schema uses two Collaboration Rules:

- **InboundEvent** – This Collaboration Rule is used by the **FileIn** e*Way's collaboration to transform the **FileInEvent** Events into **db_rcv_in** Events.
- **OutboundEvent** – This Collaboration Rule is used by the **ODBC_rcv** e*Way's collaboration to transform the **db_rcv_in** Events into **db_rcv_struct** Events.

To add the InboundEvent Collaboration Rule:

1 In the components pane of the Schema Designer, select the Collaboration Rules folder.

2 Click the  button to create a new Collaboration Rule.

3 Enter the name **InboundEvent** and click **OK**.

4 Select the newly created Collaboration Rule and click  to display the Collaboration Rule's properties.

5 In the **General** tab, select the **Pass Through** service.

6 Under the Subscriptions tab, select the **FileInEvent** Event Type.

7 Under the Publications tab, select the **db_rcv_in** Event Type.

8 Click **OK** to save and close the Collaboration Rule.

To add the OutboundEvent Collaboration Rule:

1 In the components pane of the Schema Designer, select the Collaboration Rules folder.

2 Click the  button to create a new Collaboration Rule.

3 Enter the name **OutboundEvent** and click **OK**.

4 Select the newly created Collaboration Rule and click  to display the Collaboration Rule's properties.

5 In the **General** tab, select the **Pass Through** service.

6 Under the Subscriptions tab, select the **db_rcv_in** Event Type.

7 Under the Publications tab, select the **db_rcv_struct** Event Type.

8 Click **OK** to save and close the Collaboration Rule.

4.10.8 Add and Configure the Collaborations

Each of the two e*Ways uses one Collaboration to route the Events through the sample Schema.

- **FileIn_collab** – This collaboration is used by the FileIn e*Way to process the inbound Event and queue it for the **ODBC_rcv** e*Way.
- **ODBC_rcv_collab** – This collaboration subscribes to the Event from the **FileIn_collab** and publishes the Event to the external database.

To create the FileIn_collab Collaboration:

1. In the components pane of the Schema Designer, select the **FileIn** e*Way.
2. Click the  button to create a new Collaboration.
3. Enter the name **FileIn_collab** and click **OK**.
4. Select the newly created Collaboration and click  to display the Collaboration's properties.
5. Select **InboundEvent** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **FileInEvent** Event Type and the **<External>** source.
8. Click **Add** to add a new Publication.
9. Select the **db_rcv_in** Event Type and the **ODBC_IQ** destination.
10. Click **OK** to close the Collaboration's properties.

To create the ODBC_rcv_collab Collaboration:

1. In the components pane of the Schema Designer, select the **ODBC_rcv** e*Way.
2. Click the  button to create a new Collaboration.
3. Enter the name **ODBC_rcv_collab** and click **OK**.
4. Select the newly created Collaboration and click  to display the Collaboration's properties.
5. Select **OutboundEvent** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **db_rcv_in** Event Type and the **FileIn_collab** source.
8. Click **Add** to add a new Publication.
9. Select the **db_rcv_struct** Event Type and the **<External>** destination.
10. Click **OK** to close the Collaboration's properties.

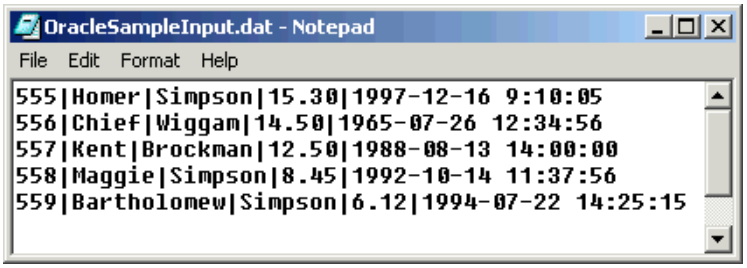
4.10.9 Run the Schema

Running the sample Schema requires a sample input file to be created. Once the input file has been created, you can start the Control Broker from a command prompt to execute the Schema. After the Schema has been run, you can use a query utility to query the results in the external database.

The sample input file

Use a text editor to create an input file to be read by the inbound file e*Way (**FileIn**). The file must be formatted to match the ETD used by the DART script. An example of an input file is shown in Figure 69. Save the file to the directory specified in the e*Way's configuration file (such as **c:\egate\data\dart**).

Figure 69 Sample Input File



To run the Control Broker:

From a command line, type the following command:

```
stccb -ln logical_name -rh registry -rs schema -un user_name -up
password
```

where

logical_name is the logical name of the Control Broker,

registry is the name of the Registry Host,

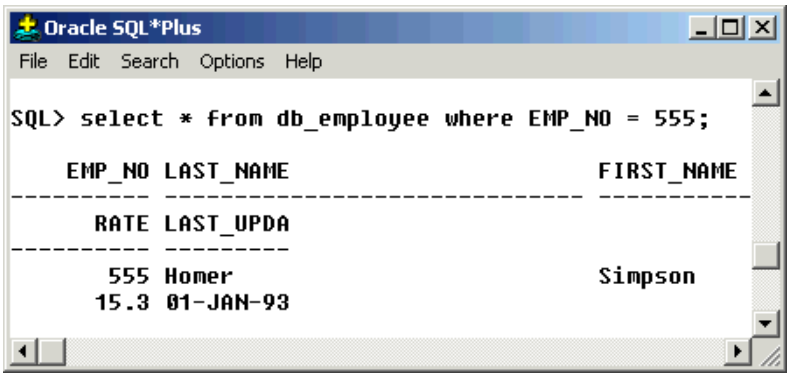
schema is the name of the Registry Schema, and

user_name and ***password*** are a valid e*Gate username/password combination.

To verify the results:

Use an SQL query utility (such as Oracle SQL Plus) to query the results of the output to the Oracle database. Figure 70 shows an example of a query to verify the results of the schema's output based on the input file used by this example.

Figure 70 Sample Output Console

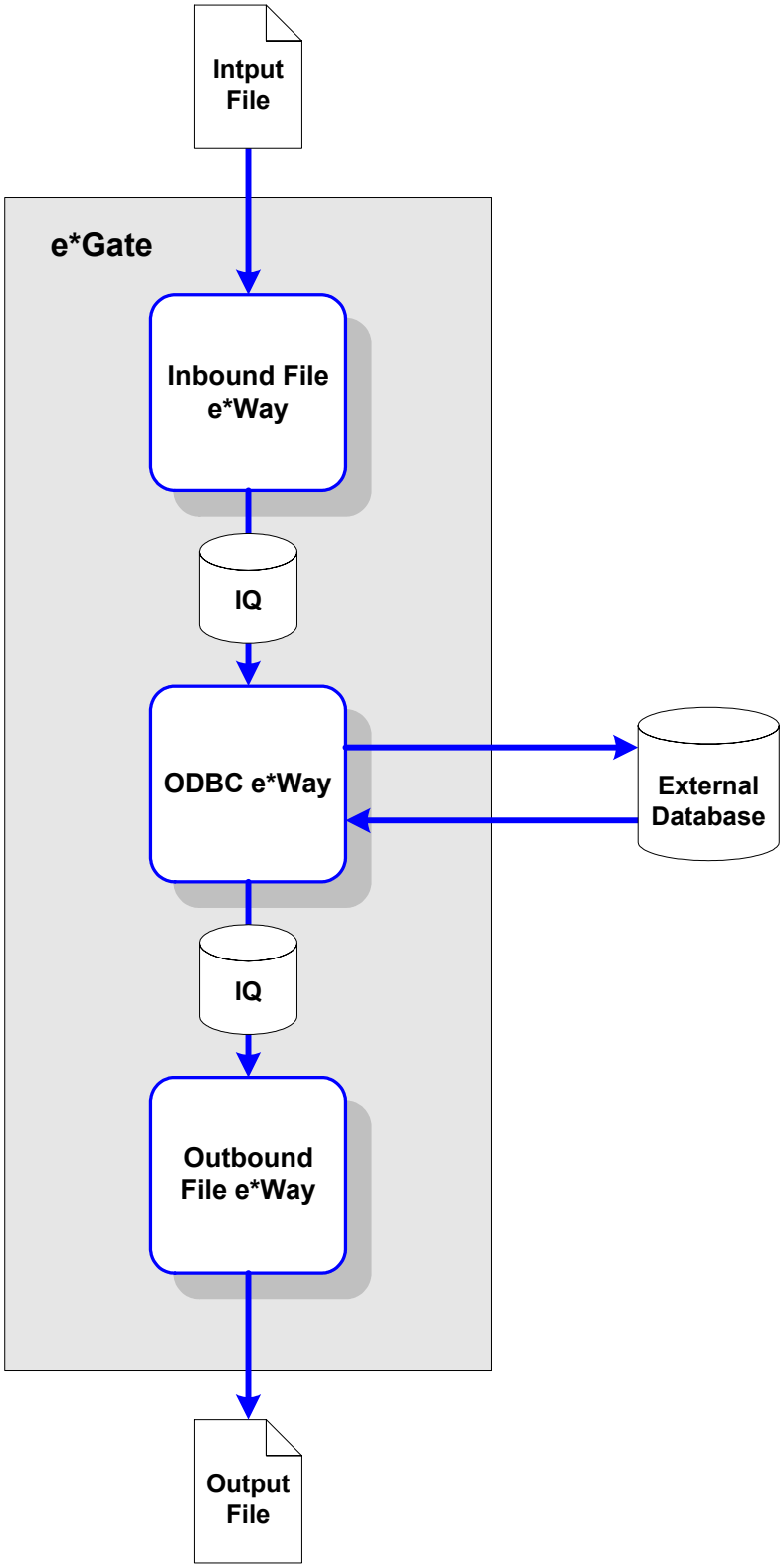


4.11 Sample Two—Polling from an ODBC Database

This section describes how to use the ODBC e*Way in a sample implementation. This sample schema demonstrates the polling of records from an external ODBC database and converting the records into e*Gate Events.

This scenario uses a file e*Way to load an input file containing employee numbers. These employee numbers are converted into e*Gate Events. The ODBC e*Way uses these inbound Events to poll employee records from the external database. As the records are returned to the ODBC e*Way, the Events are published to the outbound IQ. The Outbound file e*Way finally writes the employee records to the output file.

Figure 71 Polling from an External Database



Overview of Steps

This sample implementation follows these general steps:

Note: *The procedures outlined in this sample are not explained in the same level of detail as in* **Sample One—Publishing e*Gate Events to an ODBC Database** *on page 120. For additional information regarding the configuration of e*Gate components, see* ***Creating an End-to-End Scenario with e*Gate Integrator****.*

4.11.1 Create the Schema

The first step in deploying this sample implementation is to create a new Schema.

To add the new Schema:

1. Log into the e*Gate Schema Designer.
2. When you are prompted to select a Schema, click **New** to add a new Schema.
3. Name the Schema **ODBC_Sample2**.

4.11.2 Create the Event Type Definitions

The sample scenario requires two Event Type Definitions. The ETDs are:

- **db_request.ssc** – This ETD is used to format the inbound request Events.
- **db_reply.ssc** – This ETD is used to format the outbound reply Events.

To create the db_request ETD:

1. From the e*Gate Schema Designer, click  to launch the ETD Editor.
2. Create a new ETD named **db_request.ssc**.
3. Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 72 The **db_request.ssc** ETD

4 Save the ETD and promote it to Run Time.

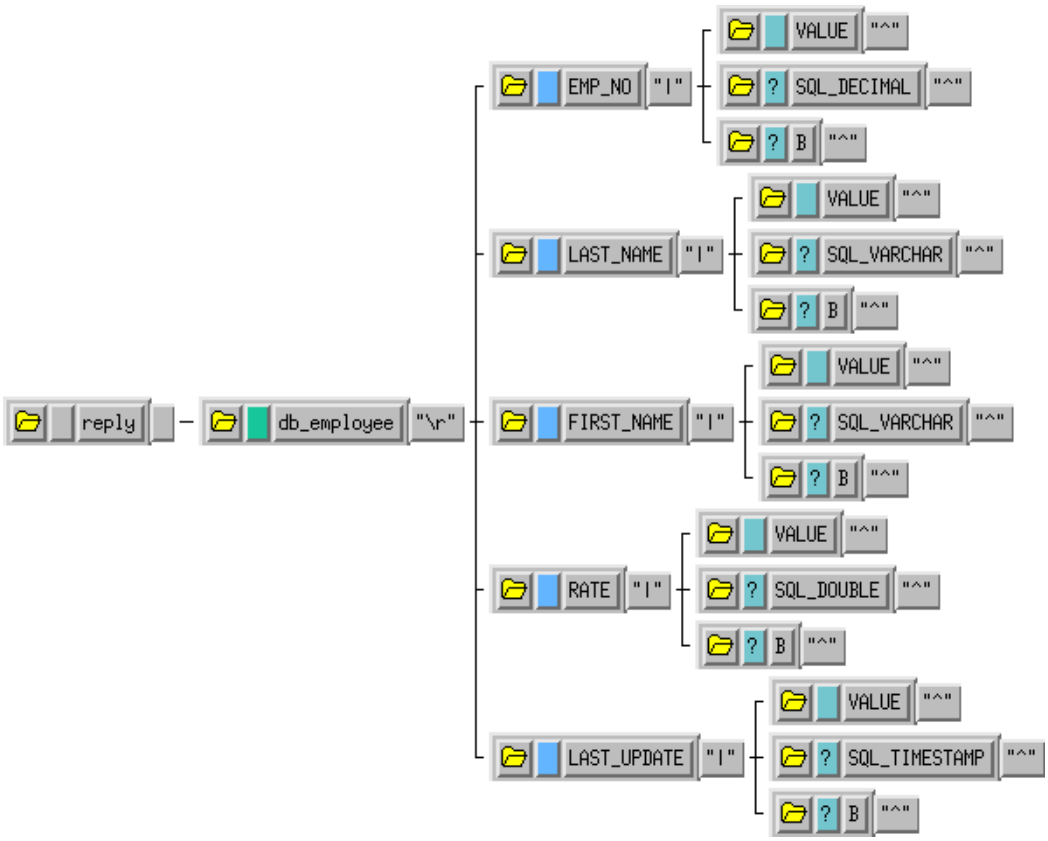
To create the db_reply ETD:

1 From the e*Gate Schema Designer, click  to launch the ETD Editor.

2 Create a new ETD named **db_reply.ssc**.

3 Add the nodes and subnodes to create an ETD with the structure shown below:

Figure 73 The **db_reply.ssc** ETD



4 Save the ETD and promote it to Run Time.

4.11.3 Add the Event Types

The sample scenario requires six Event Types. The Event Types are:

- **InboundFile** – This Event Type represents the inbound file as it is loaded from the file system.
- **InboundEvent** – This Event Type represents the inbound record that has been converted to an e*Gate Event.
- **PollRequest** – This Event Type represents the request that is sent to the external database.

- **PollReply** – This Event Type represents the reply that is returned by the external database.
- **OutboundEvent** – This Event Type represents the outbound Event to be sent to the external file system.

4.11.4 Create the Monk Scripts

This sample implementation uses a DART script (**db_poll.dsc**) to poll the external database.

To create the DART script:

1. From the e*Gate Schema Designer, click  to launch the Collaboration Rules Editor.
2. Click  to create a new DART script.

   The New Collaboration Rules Script dialog will be displayed.
3. Enter the name **db_poll** (with no file extension) as the **File name**.
4. Select **DART Poll** from the list of file types. The extension **.dsc** will be appended to the file name.
5. Click  to display the list of source files. Select **db_request.ssc** as the source file.
6. Click  to display the list of destination files. Select **db_struct.ssc** as the destination file.
7. Enter the rules as shown in Figure 74.

Note: *The rules shown in Figure 74 use a table named **db_employee**. In order for this sample to work correctly, you must either create a table in your external database called **db_employee** or change each of the references to the table name in your DART script rules as appropriate.*

Figure 74 The **db_poll.dsc** DART script

8 Click  to save the script.

9 Close the Collaboration Rules Script Editor.

4.11.5 Add and Configure the e*Ways

The sample Schema uses three e*Ways:

- **FileIn** – The **FIleIn** e*Way reads in the input data file and queues it for the ODBC e*Way.
- **ODBC** – The **ODBC** e*Way polls the db_employee table in the external database and queues the returned data for the outbound file e*Way.
- **FileOut** – The **FileOut** e*Way writes the records returned by the ODBC e*Way to the output text file.

To add and configure the FileIn e*Way:

1 In the components pane of the Schema Designer, select the Control Broker and click  to add a new e*Way.

2 Enter **FileIn** for the component name and click **OK**.

3 Select the newly created e*Way and click  to display the e*Way's properties.

4 Use the **Find** button to select **stcewfile.exe** as the executable file.

5 Click **New** to create a new configuration file.

6 Enter the parameters for the e*Way as shown in Table 14.

Table 14 FileIn e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	Yes
	AllowOutgoing	No
	Performance Testing	default
Outbound (send) settings	All settings	default
Poller (inbound) settings	PollDirectory	c:\egate\data\dart
	OutputFileName	*.dat
	AllOthers	default
Performance Testing	All settings	default

7 Save the e*Way's configuration file and promote it to run time.

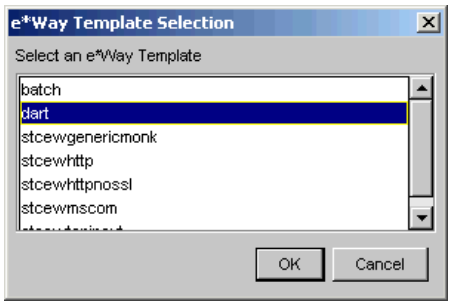
8 In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.

9 Click **OK** to save the e*Way properties.

To add and configure the ODBC e*Way:

1. In the components pane of the Schema Designer, select the Control Broker and click  to add a new e*Way.
2. Enter **ODBC** for the component name and click **OK**.
3. Select the newly created e*Way and click  to display the e*Way's properties.
4. Use the **Find** button to select **stcewgenericmonk.exe** as the executable file.
5. Click **New** to create a new configuration file.
6. Select the **dart** e*Way template and click OK. See Figure 75.

Figure 75 DART e*Way Template Selection



7. Enter the parameters for the e*Way as shown in Table 15.

Table 15 ODBC e*Way Parameters

Section Name	Parameter	Value
General Settings	All	default
Communication Setup	Start Exchange Data Schedule	Repeatedly, 30 seconds
	All others	default
Monk Configuration	Process Outgoing Message Function	monk_scripts\common\ db_poll.dsc
	All others	default
Database Setup	Database Type	ODBC
	All others	Use local settings

Note: *Use the appropriate **Database Name**, **User Name**, and **Encrypted Password** according to your local ODBC configuration.*

8. Save the e*Way's configuration file and promote it to run time.
9. In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.
10. Click **OK** to save the e*Way properties.

To add and configure the FileIn e*Way:

1. In the components pane of the Schema Designer, select the Control Broker and click  to add a new e*Way.
2. Enter **FileOut** for the component name and click **OK**.
3. Select the newly created e*Way and click  to display the e*Way's properties.
4. Use the **Find** button to select **stcewfile.exe** as the executable file.
5. Click **New** to create a new configuration file.
6. Enter the parameters for the e*Way as shown in Table 14.

Table 16 FileOut e*Way Parameters

Section Name	Parameter	Value
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
	Performance Testing	default
Outbound (send) settings	OutputDirectory	c:\egate\data\dart
	OutputFileName	PollOutput%d.dat
	All Others	default
Poller (inbound) settings	All	default
Performance Testing	All	default

7. Save the e*Way's configuration file and promote it to run time.
8. In the **Start Up** tab of the e*Way properties, select the **Start automatically** check box.
9. Click **OK** to save the e*Way properties.

4.11.6 Add the IQs

This sample Schema requires two Intelligent Queues: ODBC_1IQ and ODBC_2IQ.

To add the IQs:

1. In the components pane of the Schema Designer, select the IQ manager. Click  to create the first new IQ.
2. Enter the name **ODBC_1IQ** and click **Apply** to save the first IQ.
3. Enter the name **ODBC_2IQ** and click **OK** to save the second IQ.
4. Select the IQ Manager and click  to display the IQ Manager's properties.
5. In the **Start Up** tab of the IQ Manager's properties, select the **Start automatically** check box.

6 Click **OK** to save the IQ Manager's properties.

4.11.7 Create the Collaboration Rules

This sample schema uses four Collaboration Rules:

- **FileIn** – This Collaboration Rule is used by the **FileIn** e*Way's Collaboration to transform the **InboundFile** Events into **InboundEvent** Events.
- **ODBCRequest** – This Collaboration Rule is used by the **ODBC** e*Way's Collaboration to transform the **InboundEvent** Events into **PollRequest** Events.
- **ODBCReply** – This Collaboration Rule is used by the **ODBC** e*Way's Collaboration to transform the **PollRequest** Events into **PollReply** Events.
- **FileOut** – This Collaboration Rule is used by the **FileOut** e*Way's Collaboration to transform the **PollReply** Events into **OutboundEvent** Events.

To add the FileIn Collaboration Rule:

1 In the components pane of the Schema Designer, select the Collaboration Rules folder.

2 Click the  button to create a new Collaboration Rule.

3 Enter the name **FileIn** and click **OK**.

4 Select the newly created Collaboration Rule and click  to display the Collaboration Rule's properties.

5 In the **General** tab, select the **Pass Through** service.

6 Under the Subscriptions tab, select the **InboundFile** Event Type.

7 Under the Publications tab, select the **InboundEvent** Event Type.

8 Click **OK** to save and close the Collaboration Rule.

To add the remaining Collaboration Rules:

Follow the same steps used to add the **FileIn** Collaboration Rule using the names and Event Types shown at the beginning of this section.

4.11.8 Add and Configure the Collaborations

This sample schema uses four Collaborations:

- **FileIn_collab** – This Collaboration is used to transform the **InboundFile** Events into **InboundEvent** Events.
- **ODBCRequest_collab** – This Collaboration is used to transform the **InboundEvent** Events into **PollRequest** Events.
- **ODBCReply_collab** – This Collaboration is used to transform the **PollRequest** Events into **PollReply** Events.
- **FileOut_collab** – This Collaboration is used to transform the **PollReply** Events into **OutboundEvent** Events.

To create the FileIn_collab Collaboration:

1. In the components pane of the Schema Designer, select the **FileIn** e*Way.
2. Click the  button to create a new Collaboration.
3. Enter the name **FileIn_collab** and click **OK**.
4. Select the newly created Collaboration and click  to display the Collaboration's properties.
5. Select **InboundFile** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **InboundEvent** Event Type and the **<External>** source.
8. Click **Add** to add a new Publication.
9. Select the **InboundEvent** Event Type and the **ODBC_1IQ** destination.
10. Click **OK** to close the Collaboration's properties.

To create the ODBCRequest_collab Collaboration:

1. In the components pane of the Schema Designer, select the **ODBC** e*Way.
2. Click the  button to create a new Collaboration.
3. Enter the name **ODBCRequest_collab** and click **OK**.
4. Select the newly created Collaboration and click  to display the Collaboration's properties.
5. Select **ODBCRequest** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **InboundFile** Event Type and the **FileIn_Collab** source.
8. Click **Add** to add a new Publication.
9. Select the **ODBCRequest** Event Type and the **<External>** destination.
10. Click **OK** to close the Collaboration's properties.

To create the ODBCReply_collab Collaboration:

1. In the components pane of the Schema Designer, select the **ODBC** e*Way.
2. Click the  button to create a new Collaboration.
3. Enter the name **ODBCReply_collab** and click **OK**.
4. Select the newly created Collaboration and click  to display the Collaboration's properties.
5. Select **ODBCReply** from the list of Collaboration Rules.
6. Click **Add** to add a new Subscription.
7. Select the **ODBCRequest** Event Type and the **<External>** source.

8 Click **Add** to add a new Publication.

9 Select the **ODBCReply** Event Type and the **ODBC_2IQ** destination.

10 Click **OK** to close the Collaboration's properties.

To create the FileOut_collab Collaboration:

1 In the components pane of the Schema Designer, select the **FileOut** e*Way.

2 Click the  button to create a new Collaboration.

3 Enter the name **FileOut_collab** and click **OK**.

4 Select the newly created Collaboration and click  to display the Collaboration's properties.

5 Select **FileOut** from the list of Collaboration Rules.

6 Click **Add** to add a new Subscription.

7 Select the **ODBCReply** Event Type and the **ODBCReply_collab** source.

8 Click **Add** to add a new Publication.

9 Select the **OutboundEvent** Event Type and the **<External>** destination.

10 Click **OK** to close the Collaboration's properties.

4.11.9 Run the Schema

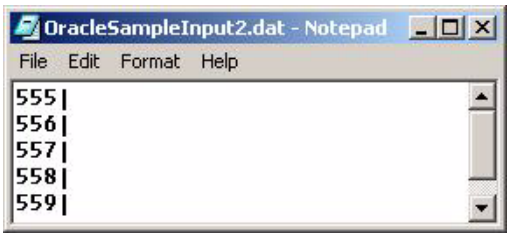
Running the sample Schema requires a sample input file to be created. Once the input file has been created, you can start the Control Broker from a command prompt to execute the Schema. After the Schema has been run, you can view the results in the output file.

The sample input file

Use a text editor to create an input file to be ready by the inbound file e*Way (**FileIn**). The file must be formatted to match the simple ETD used by the DART script (see **Figure 72 on page 132**). An example of an input file is shown in Figure 76. Save the file to the directory specified in the e*Way's configuration file (such as **c:\egate\data\dart**).

Note: *The "employee numbers" used in this example must exist in your external database. The sample shown below uses employee numbers that exist from the records in the previous sample schema.*

Figure 76 Sample Input File



To run the Control Broker:

From a command line, type the following command:

```
stccb -ln logical_name -rh registry -rs ODBC_Sample2 -un user_name
-up password
```

where

logical_name is the logical name of the Control Broker,

registry is the name of the Registry Host, and

user_name and *password* are a valid e*Gate username/password combination.

To verify the results:

Use a text editor to view the records that were written to the output file specified by the FileOut e*Way. The records should correspond to the records in the external database.

Chapter 5

JDBC/ODBC e*Way Methods

5.1 JDBC/ODBC e*Way Methods

The JDBC/ODBC e*Way contains Java methods that are used to extend the functionality of the e*Way. These methods are contained in the following classes:

5.2 com.stc.eways.jdbcx.StatementAgent Class

```
java.lang.Object
|
+ - - com.stc.eways.jdbcx.StatementAgent
```

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

PreparedStatementAgent, SQLStatementAgent, TableResultSet

```
public abstract class StatementAgent
```

extends java.lang.Object

Implements SessionEventListener, ResetEventListener

Abstract class for other Statement Agent.

Methods of the StatementAgent

resultSetTypeToString

This method gets the symbol string corresponding to the ResultSet type enumeration.

```
public static java.lang.String resultSetTypeToString(int type)
```

Name	Description
type	ResultSet type.

Returns

Enumeration symbol string.

resultSetDirToString

This method gets the symbol string corresponding to the ResultSet direction enumeration.

```
public static java.lang.String resultSetDirToString(int dir)
```

Name	Description
dir	ResultSet scroll directions.

Returns

Enumeration symbol string.

resultSetConcurToString

This method gets the symbol string corresponding to the ResultSet concurrency enumeration.

```
public static java.lang.String resultSetConcurToString(int concur)
```

Name	Description
concur	ResultSet concurrency.

Returns

Enumeration symbol string.

isClosed

This method returns the statement agent's close status.

```
public boolean isClosed()
```

Returns

True if the statement agent is closed.

queryName

This method supplies the name of the listener.

```
public java.lang.String queryName()
```

Specified By

queryName in interface SessionEventListener.

Returns

The listener's class name.

queryDescription

This method gives a description of the query.

```
public java.lang.String queryDescription()
```

Returns

The description of the query.

sessionOpen

Opens the session event handler.

```
public void sessionOpen(SessionEvent evt)
```

Specified by

sessionOpen in interface SessionEventListener

Name	Description
evt	Session event.

sessionClosed

Closes the session event handler.

```
public void sessionClosed(SessionEvent evt)
```

Specified by

sessionClosed in interface SessionEventListener

Name	Description
evt	Session event.

resetRequested

Resets the event handler.

```
public void resetRequested(ResetEvent evt)
```

Specified by

resetRequested in interface ResetEventListener

Name	Description
evt	Requested Reset event.

Throws

java.sql.SQLException

getResultSetType

Returns the result set scroll type.l

```
public int getResultSetType()
```

Returns

ResultSet type

Throws

java.sql.SQLException

getResultSetConcurrency

Returns the result set concurrency mode.

```
public int getResultSetConcurrency()
```

Returns

ResultSet concurrency

Throws

java.sql.SQLException

setEscapeProcessing

Sets escape syntax processing

```
public void setEscapeProcessing (boolean bEscape)
```

Name	Description
bEscape	True to enable False to disable

Throws

java.sql.SQLException

setCursorName

Sets result set cursor name.

```
public void setCursorName(java.lang.String sName)
```

Name	Description
sName	Cursor name.

Throws

java.sql.SQLException

setQueryTimeout

Returns query timeout duration.

```
public int getQueryTimeout()
```

Returns

The number of seconds to wait before timeout.

Throws

java.sql.SQLException

setQueryTimeout

Sets the query timeout duration

```
public void setQueryTimeout(int nInterval)
```

Name	Description
nInterval	The number of seconds before timeout.

Throws

java.sql.SQLException

getFetchDirection

Returns result set fetch direction.

```
public int getFetchDirection()
```

Returns

The fetch direction of the ResultSet: FETCH_FORWARD, FETCH_REVERSE, FETCH_UNKNOWN.

Throws

java.sql.SQLException

setFetchDirection

Sets result set fetch direction.

```
public void setFetchDirection (int iDir)
```

Name	Description
iDir	The fetch direction of the ResultSet: FETCH_FORWARD, FETCH_REVERSE, FETCH_UNKNOWN.

Throws

java.sql.SQLExeption

getFetchSize

Returns the result set prefetch record count.

```
public int getFetchSize()
```

Returns

The fetch size this StatementAgent object set.

Throws

java.sql.SQLException

getMaxRows

Returns the maximum number of fetch records.

```
public int getMaxRows()
```

Returns

The maximum number of rows that a ResultSetAgent may contain.

Throws

java.sql.SQLException

setMaxRows

Sets the maximum number of fetch records.

```
public void setMaxRows (int nRow)
```

Name	Description
nRow	The maximum number of rows in the ResultSetAgent.

Throws

java.sql.SQLException

getMaxFieldSize

Returns the maximum field data size.

```
public int getMaxFieldSize()
```

Returns

The maximum number of bytes that a ResultSetAgent column may contain; 0 means no limit.

Throws

java.sql.SQLException

setMaxFieldSize

Sets the maximum field data size.

```
public void setMaxFieldSize (int nSize)
```

Name	Description
nSize	The maximum size for a column in a ResultSetAgent.

Throws

java.sql.SQLException

getUpdateCount

Returns the records count of the last executed statement.

```
public int getUpdateCount()
```

Returns

The number of rows affected by an updated operation. O if no rows were affected or the operation was a DDL command. -1 if the result is a ResultSetAgent or there are no more results.

Throws

java.sql.SQLException

getResultSet

Returns the result set of the last executed statement.

```
public ResultSetAgent getResultSet()
```

Returns

The ResultSetAgent that was produced by the call to the method execute.

Throws

java.sql.SQLExcetpion

getMoreResults

Returns if there are more result sets.

```
public boolean getMoreResults()
```

Returns

True if the next result is a ResultSetAgent; False if it is an integer indicating an update count or there are no more results).

Throws

java.sql.SQLException

clearBatch

Clears the batch operation.

```
public void clearBatch()
```

Throws

java.sql.SQLException

executeBatch

Executes batch statements.

```
public int[] executeBatch ()
```

Returns

An array containing update counts that correspond to the commands that executed successfully. An update count of -2 means the command was successful but that the number of rows affected is unknown.

Throws

java.sql.SQLException

cancel

Cancels a statement that is being executed.

```
public void cancel()
```

Throws

java.sql.SQLException

getWarnings

Returns SQL warning object.

```
public java.sql.SQLWarning getWarnings()
```

Returns

The first SQL warning or null if there are no warnings.

Throws

java.sql.SQLException

clearWarnings

Clear all SQL Warning objects.

```
public void clearWarnings()
```

Throws

java.sql.SQLException

stmtInvoke

Invokes a method of the database Statement object of this ETD.

```
public java.lang.Object stmtInvoke (java.lang.String methodName,
java.lang.Class[] argsCls, java.lang.Object[] args)
```

Name	Description
methodName	The name of the method.
argsCls	Class array for types of formal arguments for method, in the declared order. Can be null if there are no formal arguments. However, cannot invoke constructor here.
args	Object array of formal arguments for method in the declared order. Can be null if there are no formal arguments. However, cannot invoke constructor here.

Returns

The Object instance resulting from the method invocation. Can be null if nothing is returned (void return declaration).

Throws

java.lang.Exception. Whatever exception the invoked method throws.

5.3 com.stc.eways.jdbcx.PreparedStatementAgent Class

```
java.lang.Object
|
+ --com.stc.eways.jdbcx.StatementAgent
```

```
        |
        + -- com.stc.eways.jdbcx.PreparedStatementAgent
```

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

CallableStatementAgent

```
public class PreparedStatementAgent
```

extends StatementAgent

Agent hosts PreparedStatement Object

Methods of the PreparedStatementAgent

sessionOpen

Opens the session event handler.

```
public void sessionOpen(SessionEvent evt)
```

Overrides

sessionOpen in class StatementAgent

Name	Description
evt	Session event.

setNull

Nullify value of indexed parameter.

```
public void setNull(int index, int type)
```

Name	Description
index	Parameter index starting from 1.
type	A JDBC type defined by java.sql.Types

Throws

java.sql.SQLException

setNull

Nullify value of indexed parameter.

```
public void setNul(int index, int type, java.lang.String tname)
```

Name	Description
index	Parameter index starting from 1.
type	A JDBC type defined by java.sql.Types
tname	The fully-qualified name of the parameter being set. If type is not REF, STRUCT, DISTINCT, or JAVA_OBJECT, this parameter will be ignored.

Throws

java.sql.SQLException

setObject

Sets value of indexed parameter with an object.

```
public void setObject(int index, java.lang.Object ob)
```

Name	Description
index	Parameter index starting from 1.

Name	Description
ob	An instance of a Java Object containing the input parameter value.

Throws

java.sql.SQLException

setObject

Sets value of indexed parameter with an object.

```
public void setObject(int index, java.lang.Object ob, int iType)
```

Name	Description
index	Parameter index starting from 1.
ob	An instance of a Java Object containing the input parameter value.
iType	A JDBC type defined by java.sql.Types

Throws

java.sql.SQLException

setObject

Sets value of indexed parameter with an object.

```
public void setObject(int index, java.lang.Object ob, int iType, int
iScale)
```

Name	Description
index	Parameter index starting from 1.
ob	An instance of a Java Object containing the input parameter value.
iType	A JDBC type defined by java.sql.Types
iScale	The number of digits to the right of the decimal point. Only applied to DECIMAL and NUMERIC types

Throws

java.sql.SQLException

setBoolean

Sets the boolean value of the indexed parameter.

```
public void setBoolean(int index, boolean b)
```

Name	Description
index	Parameter index starting from 1.
b	true or false.

Throws

java.sql.SQLException

setByte

Sets the byte value of the indexed parameter.

```
public void setByte(int index, byte byt)
```

Name	Description
index	Parameter index starting from 1.
byt	The byte parameter value to be set.

Throws

java.sql.SQLException

setShort

Sets the short value of the indexed parameter.

```
public void setShort(int index, short si)
```

Name	Description
index	Parameter index starting from 1.
si	The short parameter value to be set.

Throws

java.sql.SQLException

setInt

Sets the integer value of the indexed parameter.

```
public void setInt(int index, int i)
```

Name	Description
index	Parameter index starting from 1.
i	The integer parameter value to be set.

Throws

java.sql.SQLException

setLong

Sets the long value of the indexed parameter.

```
public void setLong(int index, long l)
```

Name	Description
index	Parameter index starting from 1.
l	The long parameter value to be set.

Throws

java.sql.SQLException

setFloat

Sets the float value of the indexed parameter.

```
public void setFloat(int index, float f)
```

Name	Description
index	Parameter index starting from 1.
f	The float parameter value to be set.

Throws

java.sql.SQLException

setDouble

Sets the double value of the indexed parameter.

```
public void setDouble(int index, double d)
```

Name	Description
index	Parameter index starting from 1.

Name	Description
d	The double parameter value to be set.

Throws

java.sql.SQLException

setBigDecimal

Sets the decimal value of the indexed parameter.

```
public void setBigDecimal(int index, java.math.BigDecimal dec)
```

Name	Description
index	Parameter index starting from 1.
dec	The BigDecimal parameter value to be set.

Throws

java.sql.SQLException

setDate

Sets the date value of the indexed parameter.

```
public void setDate(int index, java.sql.Date date)
```

Name	Description
index	Parameter index starting from 1.
date	The Date parameter value to be set.

Throws

java.sql.SQLException

setDate

Sets the date value of indexed parameter with time zone from calendar.

```
public void setDate(int index, java.sql.Date date, java.util.Calendar
cal)
```

Name	Description
index	Parameter index starting from 1.
date	The Date parameter value to be set.

Name	Description
cal	The calender object used to construct the date.

Throws

java.sql.SQLException

setTime

Sets the time value of the indexed parameter.

```
public void setTime(int index, java.sql.Time t)
```

Name	Description
index	Parameter index starting from 1.
t	The Time parameter value to be set.

Throws

java.sql.SQLException

setTime

Sets the time value of the indexed parameter.

```
public void setTime(int index, java.sql.Time t, java.util.Calendar
cal)
```

Name	Description
index	Parameter index starting from 1.
t	The Time parameter value to be set.
cal	The Calendar object used to construct the time.

Throws

java.sql.SQLException

setTimestamp

Sets the timestamp value of the indexed parameter.

```
public void setTimestamp(int index, java.sql.Timestamp ts)
```

Name	Description
index	Parameter index starting from 1.
ts	The Timestamp parameter value to be set.

Throws

java.sql.SQLException

setTimestamp

Sets the timestamp value of the indexed parameter with the time zone from the calendar.

```
public void setTimestamp(int index, java.sql.timestamp ts,
java.util.Calendar cal)
```

Name	Description
index	Parameter index starting from 1.
ts	The Timestamp parameter value to be set.
cal	The Calendar object used to construct the timestamp.

Throws

java.sql.SQLException

setString

Sets the string value of the indexed parameter.

```
public void setString(int index, java.lang.String s)
```

Name	Description
index	Parameter index starting from 1.
s	The String parameter value to be set.

Throws

java.sql.SQLException

setBytes

Sets the byte array value of the indexed parameter.

```
public void setBytes(int index, byte[] bytes)
```

Name	Description
index	Parameter index starting from 1.
bytes	The byte array parameter value to be set.

Throws

java.sql.SQLException

setAsciiStream

Sets the character value of the indexed parameter with an input stream and specified length.

```
public void setAsciiStream(int index, java.io.InputStream is, int
length)
```

Name	Description
index	Parameter index starting from 1.
is	The InputStream that contains the Ascii parameter value to be set.
length	The number of bytes to be read from the stream and sent to the database.

Throws

java.sql.SQLException

setBinaryStream

Sets the binary value of the indexed parameter with an input stream and specified length.

```
public void setBinaryStream(int index, java.io.InputStream is, int
length)
```

Name	Description
index	Parameter index starting from 1.
is	The InputStream that contains the binary parameter value to be set.
length	The number of bytes to be read from the stream and sent to the database.

Throws

java.sql.SQLException

setCharacterStream

Sets the character value of the indexed parameter with a reader stream and specified length.

```
public void setCharacterStream(int index, java.io.Reader rd, int
length)
```

Name	Description
index	Parameter index starting from 1.
rd	The Reader that contains the Unicode parameter value to be set.
length	The number of characters to be read from the stream and sent to the database.

Throws

java.sql.SQLException

setArray

Sets the Array value of the indexed parameter.

```
public void setArray(int index, java.sql.Array a)
```

Name	Description
index	Parameter index starting from 1.
a	The Array value to be set.

Throws

java.sql.SQLException

setBlob

Sets the Blob value of the indexed parameter.

```
public void setBlob(int index, java.sql.Blob blob)
```

Name	Description
index	Parameter index starting from 1.
blob	The Blob value to be set.

Throws

java.sql.SQLException

setClob

Sets the Clob value of the indexed parameter.

```
public void setClob(int index, java.sql.Clob clob)
```

Name	Description
index	Parameter index starting from 1.
clob	The Clob value to be set.

Throws

java.sql.SQLException

setRef

Sets the Ref value of the indexed parameter.

```
public void setRef(int index, java.sql.Ref ref)
```

Name	Description
index	Parameter index starting from 1.
ref	The Ref parameter value to be set.

Throws

java.sql.SQLException

clearParameters

Clears the parameters of all values.

```
public void clearParameters()
```

Throws

java.sql.SQLException

addBatch

Adds a set of parameters to the list of commands to be sent as a batch.

```
public void addBatch()
```

Throws

java.sql.SQLException

execute

Executes the Prepared SQL statement.

```
public void execute()
```

Throws

java.sql.SQLException

executeQuery

Executes the prepared SQL query and returns a ResultSetAgent that contains the generated result set.

```
public ResultSetAgent executeQuery()
```

Returns

ResultSetAgent or null.

Throws

java.sql.SQLException

executeUpdate

Executes the prepared SQL statement and returns the number of rows that were affected.

```
public int executeUpdate()
```

Returns

The number of rows affected by the update operation; 0 if no rows were affected.

Throws

java.sql.SQLException

5.4 com.stc.eways.jdbcx.PreparedStatementResultSet Class

java.lang.Object

|

+ -- **com.stc.eways.jdbcx.PreparedStatementResultSet**

```
public abstract class PreparedStatementResultSet
```

extends java.lang.Object

Base class for Result Set returned from a Prepared Statement execution.

Constructors of PreparedStatementResultSet

PreparedStatementResultSet

Methods of PreparedStatementResultSet

isLast on page 171
last on page 170
next on page 169
previous on page 169
refreshRow on page 209
relative on page 170
getFetchDirection on page 167
getFetchSize on page 168
updateRow on page 189
wasNull on page 188

Constructor PreparedStatementResultSet

Constructs a Prepared Statement Result Set object.

```
public PreparedStatementResultSet(ResultSetAgent rsAgent)
```

Name	Description
rsAgent	The ResultSetAgent underlying control.

getMetaData

Retrieves a ResultSetMetaData object that contains ResultSet properties.

```
public java.sql.ResultSetMetaData getMetaData()
```

Returns

ResultSetMetaData object

Throws

java.sql.SQLException

getConcurrency

Gets the concurrency mode for this ResultSet object.

```
public int getConcurrency()
```

Returns

Concurrency mode

Throws

java.sql.SQLException

getFetchDirection

Gets the direction suggested to the driver as the row fetch direction.

```
public int getFetchDirection()
```

Returns

Row fetch direction

Throws

java.sql.SQLException

setFetchDirection

Gives the driver a hint as to the row process direction.

```
public void setFetchDirection(int iDir)
```

Name	Description
iDir	Fetch direction to use.

Throws

java.sql.SQLException

getFetchSize

Gets the number of rows to fetch suggested to the driver.

```
public int getFetchSize()
```

Returns

Number of rows to fetch at a time.

Throws

java.sql.SQLException

setFetchSize

Gives the drivers a hint as to the number of rows that should be fetched each time.

```
public void setFetchSize(int nSize)
```

Name	Description
nSize	Number of rows to fetch at a time.

Throws

java.sql.SQLException

getCursorName

Retrieves the name for the cursor associated with this ResultSet object.

```
public java.lang.String getCursorName()
```

Returns

Name of cursor

Throws

java.sql.SQLException

---

close

Immediately releases a ResultSet object's resources.

```
public void close()
```

Throws

java.sql.SQLException

---

next

Moves the cursor to the next row of the result set.

```
public boolean next()
```

Returns

true if successful

Throws

java.sql.SQLException

---

previous

Moves the cursor to the previous row of the result set.

```
public boolean previous()
```

Returns

true if successful

Throws

java.sql.SQLException

---

absolute

Moves the cursor to the specified row of the result set.

```
public boolean absolute(int index)
```

Returns

true if successful

Throws

java.sql.SQLException

relative

Moves the cursor to the specified row relative to the current row of the result set.

```
public boolean relative(int index)
```

Returns

true if successful

Throws

java.sql.SQLException

first

Moves the cursor to the first row of the result set.

```
public boolean first()
```

Returns

true if successful

Throws

java.sql.SQLException

isFirst

Determines whether the cursor is on the first row of the result set.

```
public boolean isFirst()
```

Returns

true if on the first row.

Throws

java.sql.SQLException

last

Moves the cursor to the last row of the result set.

```
public boolean last()
```

Returns

true if successful

Throws

java.sql.SQLException

isLast

Determines whether the cursor is on the last row of the result set.

```
public boolean isLast()
```

Returns

true if on the last row

Throws

java.sql.SQLException

beforeFirst

Moves the cursor before the first row of the result set.

```
public void beforeFirst()
```

Throws

java.sql.SQLException

isBeforeFirst

Determines whether the cursor is before the first row of the result set.

```
public boolean isBeforeFirst()
```

Returns

true if before the first row

Throws

java.sql.SQLException

afterLast

Moves the cursor after the last row of the result set.

```
public void afterLast()
```

Throws

java.sql.SQLException

isAfterLast

Determines whether the cursor is after the last row of the result set.

```
public boolean isAfterLast()
```

Returns

true if after the last row

Throws

java.sql.SQLException

getType

Retrieves the scroll type of cursor associated with the result set.

```
public int getType()
```

Returns

Scroll type of cursor.

Throws

java.sql.SQLException

findColumn

Returns the column index for the named column in the result set.

```
public int findColumn(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Corresponding column index.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column.

```
public java.lang.Object getObject(int index)
```

Name	Description
index	Column index.

Returns

Object form of column value.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column.

```
public java.lang.Object getObject(java.lang.String index)
```

Name	Description
index	Column index.

Returns

Object form of column value.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column using the given type map.

```
public java.lang.Object getObject(int index, java.util.Map.map)
```

Name	Description
index	Column index.
map	Type map.

Returns

Object form of column value.

Throws

java.sql.SQLException

getObject

Gets the object value of the specified column using the given type map.

```
public java.lang.Object getObject(java.lang.String index,
java.util.Map map)
```

Name	Description
index	Column index.
map	Type map.

Returns

Object form of column value.

Throws

java.sql.SQLException

---

getBoolean

Gets the boolean value of the specified column.

```
public boolean getBoolean(int index)
```

Name	Description
index	Column index.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

---

getBoolean

Gets the boolean value of the specified column.

```
public boolean getBoolean(java.lang.String index))
```

Name	Description
index	Column name.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

---

getByte

Gets the byte value of the specified column.

```
public byte getByte(int index)
```

Name	Description
index	Column index.

Returns

Boolean value of the column.

Throws

java.sql.SQLException

---

getShort

Gets the short value of the specified column.

```
public short getShort(int index)
```

Name	Description
index	Column index.

Returns

Short value of the column.

Throws

java.sql.SQLException

---

getShort

Gets the short value of the specified column.

```
public short getShort(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Short value of the column.

Throws

java.sql.SQLException

---

getInt

Gets the integer value of the specified column.

```
public int getInt(int index)
```

Name	Description
index	Column index.

Returns

Int value of the column.

Throws

java.sql.SQLException

getInt

Gets the integer value of the specified column.

```
public int getInt(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Int value of the column.

Throws

java.sql.SQLException

getLong

Gets the long value of the specified column.

```
public long getLong(int index)
```

Name	Description
index	Column index.

Returns

Long value of the column.

Throws

java.sql.SQLException

getLong

Gets the long value of the specified column.

```
public long getLong(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Long value of the column.

Throws

java.sql.SQLException

getFloat

Gets the float value of the specified column.

```
public float getFloat(int index)
```

Name	Description
index	Column index.

Returns

Float value of the column.

Throws

java.sql.SQLException

getFloat

Gets the float value of the specified column.

```
public float getFloat(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Float value of the column.

Throws

java.sql.SQLException

getDouble

Gets the double value of the specified column.

```
public double getDouble(int index)
```

Name	Description
index	Column index.

Returns

Double value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the decimal value of the specified column.

```
public java.math.BigDecimal getBigDecimal(int index)
```

Name	Description
index	Column index.

Returns

Big decimal value of the column.

Throws

java.sql.SQLException

getBigDecimal

Gets the decimal value of the specified column.

```
public java.math.BigDecimal getBigDecimal(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Big decimal value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column.

```
public java.sql.Date getDate(int index)
```

Name	Description
index	Column index.

Returns

Date value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column.

```
public java.sql.Date getDate(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Date value of the column.

Throws

java.sql.SQLException

getDate

Gets the date value of the specified column using the time zone from the calendar.

```
public java.sql.Date getDate(java.lang.String index,
java.util.Calendar calendar)
```

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

Date value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(int index)
```

Name	Description
index	Column index.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(int index, java.util.Calendar calendar)
```

Name	Description
index	Column index.
calendar	Calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTime

Gets the time value of the specified column.

```
public java.sql.Time getTime(java.lang.String index,
java.util.Calendar calendar)
```

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

Time value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column.

```
public java.sql.Timestamp getTimestamp(int index)
```

Name	Description
index	Column index.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column.

```
public java.sql.Timestamp getTimestamp(java.lang.String index)
```

Name	Description
index	Column name.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column using the time zone from the calendar.

```
public java.sql.Timestamp getTimestamp(int index, java.util.Calendar
calendar)
```

Name	Description
index	Column index.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getTimestamp

Gets the timestamp value of the specified column using the time zone from the calendar.

```
public java.sql.Timestamp getTimestamp(java.lang.String index,
java.util.Calendar calendar)
```

Name	Description
index	Column name.
calendar	Calendar to use.

Returns

The timestamp value of the column.

Throws

java.sql.SQLException

getString

Gets the string value of the specified column.

```
public java.lang.String getString(int index)
```

Name	Description
index	Column index.

Returns

Returns the String value of the column.

Throws

java.sql.SQLException

---

getString

Gets the string value of the specified column.

```
public java.lang.String getString(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Returns the String value of the column.

Throws

java.sql.SQLException

---

getBytes

Gets the byte array value of the specified column.

```
public byte[] getBytes(int index)
```

Name	Description
index	Column index.

Returns

Byte array value of the column.

Throws

java.sql.SQLException

---

getBytes

Gets the byte array value of the specified column.

```
public byte[] getBytes(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Byte array value of the column.

Throws

java.sql.SQLException

getAsciiStream

Retrieves the value of the specified column value as a stream of ASCII characters.

```
public java.io.InputStream getAsciiStream(int index)
```

Name	Description
index	Column index.

Returns

ASCII output stream value of the column.

Throws

java.sql.SQLException

getAsciiStream

Retrieves the value of the specified column value as a stream of ASCII characters.

```
public java.io.InputStream getAsciiStream(java.lang.String index)
```

Name	Description
index	Column name.

Returns

ASCII output stream value of the column.

Throws

java.sql.SQLException

getBinaryStream

Retrieves the value of the specified column as a stream of uninterpreted bytes.

```
public java.io.InputStream getBinaryStream(int index)
```

Name	Description
index	Column index.

Returns

Binary out steam value of the column.

Throws

java.sql.SQLException

getBinaryStream

Retrieves the value of the specified column as a stream of uninterpreted bytes.

```
public java.io.InputStream getBinaryStream(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Binary out steam value of the column.

Throws

java.sql.SQLException

getCharacterStream

Retrieves the value of the specified column as a Reader object.

```
public java.io.Reader getCharacterStream(int index)
```

Name	Description
index	Column index.

Returns

Reader for value in the column.

Throws

java.sql.SQLException

getArray

Gets the Array value of the specified column.

```
public java.sql.Array getArray(int index)
```

Name	Description
index	Column index.

Returns

Array value of the column.

Throws

java.sql.SQLException

getBlob

Gets the Blob value of the specified column.

```
public java.sql.Blob getBlob(int index)
```

Name	Description
index	Column index.

Returns

Blob value of the column.

Throws

java.sql.SQLException

getBlob

Gets the Blob value of the specified column.

```
public java.sql.Blob getBlob(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Blob value of the column.

Throws

java.sql.SQLException

getClob

Gets the Clob value of the specified column.

```
public java.sql.Clob getClob(int index)
```

Name	Description
index	Column index.

Returns

Clob value of the column.

Throws

java.sql.SQLException

getClob

Gets the Clob value of the specified column.

```
public java.sql.Clob getClob(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Clob value of the column.

Throws

java.sql.SQLException

getRef

Gets the Ref value of the specified column.

```
public java.sql.Ref getRef(int index)
```

Name	Description
index	Column index.

Returns

Ref value of the column.

Throws

java.sql.SQLException

getRef

Gets the Ref value of the specified column.

```
public java.sql.Ref getRef(java.lang.String index)
```

Name	Description
index	Column name.

Returns

Ref value of the column.

Throws

java.sql.SQLException

---

wasNull

Checks to see if the last value read was SQL NULL or not.

```
public boolean wasNull()
```

Returns

true if SQL NULL.

Throws

java.sql.SQLException

---

getWarnings

Gets the first SQL Warning that has been reported for this object.

```
public java.sql.SQLWarning getWarnings()
```

Returns

SQL warning.

Throws

java.sql.SQLException

---

clearWarnings

Clears any warnings reported on this object.

```
public void clearWarnings()
```

Throws

java.sql.SQLException

---

getRow

Retrieves the current row number in the result set.

```
public int getRow()
```

Returns

Current row number

Throws

java.sql.SQLException

refreshRow

Replaces the values int the current row of the result set with their current values in the database.

```
public void refreshRow()
```

Throws

java.sql.SQLException

insertRow

Inserts the contents of the insert row into the result set and the database.

```
public void insertRow()
```

Throws

java.sql.SQLException

updateRow

Updates the underlying database with the new contents of the current row.

```
public void updateRow()
```

Throws

java.sql.SQLException

deleteRow

Deletes the current row from the result set and the underlying database.

public void deleteRow()

Throws

java.sql.SQLException

5.5 com.stc.eways.jdbcx.SqlStatementAgent Class

java.lang.Object

|

+ -- com.stc.eways.jdbcx.StatementAgent

|

+ -- **com.stc.eways.jdbcx.SqlStatementAgent**

All Implemented Interfaces

ResetEventListener, SessionEventListener

public class SqlStatementAgent

extends StatementAgent

SQLStatement Agent that hosts a managed Statement object.

Constructors of the SqlStatementAgent

SqlStatementAgent

SqlStatementAgent

Methods of the SqlStatementAgent

addBatch on page 192
execute on page 191
executeQuery on page 191
executeUpdate on page 191

Constructor SqlStatementAgent

Creates new SQLStatementAgent with scroll direction TYPE_FORWARD_ONLY and concurrency CONCUR_READ_ONLY.

```
public SqlStatementAgent(Session session)
```

Name	Description
session	Connection session.

Constructor SqlStatementAgent

Creates a new SQLStatementAgent.

```
public SqlStatementAgent(Session session, int iScroll, int iConcur)
```

Name	Description
session	Connection session.
iScroll	Scroll direction: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE.
iConcur	Concurrency: CONCUR_READ_ONLY, CONCUR_UPDATABLE.

execute

Executes the specified SQL statement.

```
public boolean execute(java.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Returns

true if the first result is a ResultSetAgent or false if it is an integer.

Throws

java.sql.SQLException

executeQuery

Executes the specified SQL query and returns a ResultSetAgent that contains the generated result set.

```
public ResultSetAgent executeQuery(java.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Returns

A ResultSetAgent or null

Throws

java.sql.SQLException

executeUpdate

Executes the specified SQL statement and returns the number of rows that were affected.

```
public int executeUpdate(jave.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Returns

The number of rows affected by the update operation; 0 if no rows were affected.

Throws

java.sql.SQLException

addBatch

Adds the specified SQL statement to the list of commands to be sent as a batch.

```
public void addBatch(java.lang.String sSql)
```

Name	Description
sSql	SQL statement.

Throws

java.sql.SQLException

5.6 com.stc.eways.jdbcx.CallableStatementAgent Class

java.lang.Object

|

+ -- com.stc.eways.jdbcx.StatementAgent

|

+ -- com.stc.eways.jdbcx.PreparedStatementAgent

|

+ -- **com.stc.eways.jdbcx.CallableStatementAgent**

All Implemented Interfaces

ResetEventListener, SessionEventListener

Direct Known Subclasses

StoredProcedureAgent

```
public abstract class CallableStatementAgent
```

extends PreparedStatementAgent

Agent hosts CallableStatement interface

Constructors of the CallableStatementAgent

CallableStatementAgent

CallableStatementAgent

CallableStatementAgent

Methods of the CallableStatementAgent

Constructor CallableStatementAgent

Creates new CallableStatementAgent with scroll direction TYPE_FORWARD_ONLY and concurrency CONCUR_READ_ONLY.

```
public CallableStatementAgent(Session session, java.lang.String
sCommand)
```

Name	Description
session	Connection session.
sCommand	The Call statement used to invoke a stored procedure.

Constructor CallableStatementAgent

Creates a new CallableStatementAgent.

```
public CallableStatementAgent(Session session, int iScroll, int
iConcur)
```

Name	Description
session	Connection session.
iScroll	Ignored.
iConcur	Ignored

Constructor CallableStatement Agent

Creates a new CallableStatementAgent.

```
public CallableStatementAgent(Session session, java.lang.String
sCommand, int iScroll, int iConcur)
```

Name	Description
session	Connection session.
sCommand	The Call statement used to invoke a stored procedure.
iScroll	Scroll direction: TYPE_FORWARD_ONLY, TYPE SCROLL_INSENSITIVE, TYPE_SCROLL_SENSITIVE
iConcur	Concurrency: CONCUR_READ_ONLY, CONCUR_UPDATEABLE

sessionOpen

Opens the session event handler.

```
public void sessionOpen(SessionEvent evt)
```

Overrides

sessionOpen in class PreparedStatementAgent

Name	Description
evt	Session event.

registerOutParameter

Registers the indexed OUT parameter with specified type.

```
public void registerOutParameter(int index, int iType)
```

Name	Description
index	Parameter index starting from 1.
iType	A JDBC type defined by java.sql.Types.

Throws

java.sql.SQLException

registerOutParameter

Registers the indexed OUT parameter with specified type and scale.

```
public void registerOutParameter(int index, int iType, int iScale)
```

Name	Description
index	Parameter index starting from 1.
iType	A JDBC type defined by java.sql.Types.
iScale	The number of digits to the right of the decimal point. Only applied to DECIMAL and NUMERIC types.

Throws

java.sql.SQLException

registerOutParameter

Registers the indexed OUT parameter with specified user-named type or REF type.

```
public void registerOutParameter(int index, int iType,
java.lang.String sType)
```

Name	Description
index	Parameter index starting from 1.
iType	A JDBC type defined by java.sql.Types.
tName	The fully-qualified name of the parameter being set. It is intended to be used by REF, STRUCT, DISTINCT, or JAVA_OBJECT.

Throws

java.sql.SQLException

wasNull

Returns whether or not the last OUT parameter read had the SQL NULL value.

```
public boolean wasNull()
```

Returns

true if the parameter read is SQL NULL; otherwise, false

Throws

java.sql.SQLException

getObject

Gets the value of the indexed parameter as an instance of Object.

```
public java.lang.Object getObject(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

The Object value

Throws

java.sql.SQLException

getObject

Gets the value of the indexed parameter as an instance of Object and uses map for the customer mapping of the parameter value.

```
public java.lang.Object getObject(int index, java.util.Map map)
```

Name	Description
index	Parameter index starting from 1.
map	A Map object for mapping from SQL type names for user-defined types to classes in the Java programming language.

Returns

An Object value

Throws

java.sql.SQLException

getBoolean

Gets the boolean value of the indexed parameter.

```
public boolean getBoolean(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A boolean value

Throws

java.sql.SQLException

getByte

Gets byte value of the indexed parameter.

```
public byte getByte(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A byte value

Throws

java.sql.SQLException

getShort

Gets short value of the indexed parameter.

```
public short getShort(int index)
```

Returns

A short value

Throws

java.sql.SQLException

getInt

Gets integer value of the indexed parameter.

```
public int getInt(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A int value

Throws

java.sql.SQLException

getLong

Gets long value of the indexed parameter.

```
public long getLong(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A long value

Throws

java.sql.SQLException

getFloat

Gets float value of the indexed parameter.

```
public float getFloat(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A float value

Throws

java.sql.SQLException

getDouble

Gets double value of the indexed parameter.

```
public double getDouble(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A float value

Throws

java.sql.SQLException

getBigDecimal

Gets decimal value of the indexed parameter.

```
public java.math.BigDecimal getBigDecimal(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A BigDecimal object

Throws

java.sql.SQLException

getDate

Gets date value of the indexed parameter.

```
public java.sql.Date getDate(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Date object

Throws

java.sql.SQLException

getDate

Gets date value of the indexed parameter with time zone from calendar.

```
public java.sql.Date getDate(int index, java.util.Calendar calendar)
```

Name	Description
index	Parameter index starting from 1.
cal	The Calendar object used to construct the timestamp.

Returns

A Date object

Throws

java.sql.SQLException

getTime

Gets time value of the indexed parameter.

```
public java.sql.Time getTime(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Time object

Throws

java.sql.SQLException

getTime

Gets time value of the indexed parameter with time zone from calendar.

```
public java.sql.Time getTime(int index, java.util.Calendar calendar)
```

Name	Description
index	Parameter index starting from 1.
cal	The Calendar object used to construct the timestamp.

Returns

A Time object

Throws

java.sql.SQLException

getTimestamp

Gets timestamp value of the indexed parameter.

```
public java.sql.timestamp getTimestamp(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Timestamp object

Throws

java.sql.SQLException

getTimestamp

Gets timestamp value of the indexed parameter.

```
public java.sql.timestamp getTimestamp(int index, java.util.Calendar
calendar)
```

Name	Description
index	Parameter index starting from 1.
cal	The Calendar object used to construct the timestamp.

Returns

A Timestamp object

Throws

java.sql.SQLException

getString

Gets string value of the indexed parameter.

```
public java.lang.String getString(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A String object

Throws

java.sql.SQLException

getBytes

Gets byte array value of the indexed parameter.

```
public byte[] getBytes(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

An array of bytes

Throws

java.sql.SQLException

getArray

Gets Array value of the indexed parameter.

```
public java.sql.Array getArray(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

An Array object

Throws

java.sql.SQLException

getBlob

Gets Blob value of the indexed parameter.

```
public java.sql.Blob getBlob(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Blob object

Throws

java.sql.SQLException

getClob

Gets Clob value of the indexed parameter.

```
public java.sql.Clob getClob(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Blob object

Throws

java.sql.SQLException

getRef

Gets Ref value of the indexed parameter.

```
public java.sql.Ref getRef(int index)
```

Name	Description
index	Parameter index starting from 1.

Returns

A Ref object

Throws

java.sql.SQLException

5.7 com.stc.eways.jdbcx.TableResultSet Class

java.lang.Object

|

+ -- com.stc.eways.jdbcx.StatementAgent

|

+ -- **com.stc.eways.jdbcx.TableResultSet**

All Implemented Interfaces

ResetEventListener, SessionEventListener

```
public abstract class TableResultSet
```

extends StatementAgent

ResultSet to map selected records of table in the database

Methods of the TableResultSet

select

Select table records.

```
public void select(java.lang.String sWhere)
```

Name	Description
sWhere	Where condition for the query.

Throws

java.sql.SQLException

next

Navigate one row forward.

```
public boolean next()
```

Returns

true if the move to the next row is successful; otherwise, false.

Throws

java.sql.SQLException

previous

Navigate one row backward. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean previous()
```

Returns

true if the cursor successfully moves to the previous row; otherwise, false.

Throws

java.sql.SQLException

absolute

Move cursor to specified row number. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

Name	Description
row	An integer other than 0.

Returns

true if the cursor successfully moves to the specified row; otherwise, false.

Throws

java.sql.SQLException

relative

Move the cursor forward or backward a specified number of rows. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean relative(int rows)
```

Name	Description
rows	The number of rows to move the cursor, starting at the current row. If the rows are positive, the cursor moves forward; if the rows are negative, the cursor moves backwards.

Returns

true if the cursor successfully moves to the number of rows specified; otherwise, false.

Throws

java.sql.SQLException

first

Move the cursor to the first row of the result set. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean first()
```

Returns

true if the cursor successfully moves to the first row; otherwise, false.

Throws

java.sql.SQLException

isFirst

Check if the cursor is on the first row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isFirst()
```

Returns

true if the cursor successfully moves to the first row; otherwise, false.

Throws

java.sql.SQLException

last

Move to the last row of the result set. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean last()
```

Returns

true if the cursor successfully moves to the last row; otherwise, false.

Throws

java.sql.SQLException

isLast

Check if the cursor is positioned on the last row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isLast()
```

Returns

true if the cursor is on the last row; otherwise, false

Throws

java.sql.SQLException

beforeFirst

Move the cursor before the first row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public void beforeFirst()
```

Throws

java.sql.SQLException

isBeforeFirst

Check if the cursor is positioned before the first row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isBeforeFirst()
```

Returns

true if the cursor successfully moves before the first row; otherwise, false

Throws

java.sql.SQLException

afterLast

Move the cursor after the last row.It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public void afterLast()
```

Throws

java.sql.SQLException

isAfterLast

Returns true if the cursor is positioned after the last row. It should be called only on ResultSetAgent objects that are TYPE_SCROLL_SENSITIVE or TYPE_SCROLL_INSENSITIVE.

```
public boolean isAfterLast()
```

Returns true if the cursor successfully moves after the last row; otherwise, false.

Throws

java.sql.SQLException

findColumn

Finds the index of the named column.

public int findColumn(java.lang.String index)

Throws

java.sql.SQLException

getAsciiStream

Returns the column data as an AsciiStream.

```
public java.io.InputStream getAsciiStream(int index)
```

Throws

java.sql.SQLException

getAsciiStream

Returns the column data as an AsciiStream.

```
public java.io.InputStream getAsciiStream(java.lang.String
columnName)
```

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as BinaryStream.

```
public java.io.InputStream getBinaryStream(int index)
```

Throws

java.sql.SQLException

getBinaryStream

Returns the column data as BinaryStream.

```
public java.io.InputStream getBinaryStream(java.lang.String
columnName)
```

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as CharacterStream.

```
public java.io.Reader getCharacterStream(int index)
```

Throws

java.sql.SQLException

getCharacterStream

Returns the column data as CharacterStream.

```
public java.io.Reader getCharacterStream(java.lang.String columnName)
```

Throws

java.sql.SQLException

refreshRow

Refreshes the current row with its most recent value from the database.

```
public void refreshRow()
```

Throws

java.sql.SQLException

insertRow

Inserts the contents of the current row into the database.

```
public void insertRow()
```

Throws

java.sql.SQLException

updateRow

Updates the contents of the current row into the database.

```
public void updateRow()
```

Throws

java.sql.SQLException

deleteRow

Deletes the contents of the current row from the database.

```
public void deleteRow()
```

Throws

java.sql.SQLException

moveToInsertRow

Moves the current position to a new insert row.

```
public void moveToInsertRow()
```

Throws

java.sql.SQLException

moveToCurrentRow

Moves the current position to the current row. It is used after you insert a row.

```
public void moveToCurrentRow()
```

Throws

java.sql.SQLException

cancelRowUpdates

Cancels any updates made to this row.

```
public void cancelRowUpdates()
```

Throws

java.sql.SQLException

rowInserted

Returns true if the current row has been inserted.

```
public boolean rowInserted()
```

Throws

java.sql.SQLException

rowUpdated

Returns true i the current row has been updated.

```
public boolean rowUpdated()
```

Throws

java.sql.SQLException

rowDeleted

Returns true if the current row has been deleted.

```
public boolean rowDeleted()
```

Throws

java.sql.SQLException

wasNull

Returns true if the last data retrieved is NULL.

```
public boolean wasNull()
```

Throws

java.sql.SQLException

5.8 $DB Configuration Node Methods

The following methods are associated with the $DB configuration node in the Collaboration. These methods are driver and database specific and will vary from database to database. It is recommended that you consult your specific databases documentation.

These methods are contained in the following classes:

- **Com_stc_jdbcx_dbcfg.DataSource** on page 211
- **Com_stc_jdbcx_dbcfg** on page 220

5.9 Com_stc_jdbcx_dbcfg.DataSource

Java.lang.Object

|

+ - - com_stc_jdbcx_dbcfg.Com_stc_jdbcx_dbcfg.DataSource

Direct Known Subclasses

```
public class Com_stc_jdbcx_dbcfg.DataSource
```

extends java.lang.Object

Methods of the DataSource

addDataSourceAttributes on page 217	**addDataSourceAttributes** on page 217
clearDataSourceAttributes on page 218	**countDataSourceAttributes** on page 217

getClass

Retrieves the class name.

```
public java.lang.String getClass_()
```

Returns

Class name.

setClass

Specifies the name of the Java class that implements the JDBC driver, the Connection Pool DataSource or the XA DataSource.

The selection of the class should match the connection method that is used. For example, driver class should use the “URL” connection method. Connection Pool DataSource should use “Pooled Data Source” method.

By default, it is set to the Sun JDBC/ODBC bridge driver. e.g. sun.jdbc.odbc.JdbcOdbcDriver.

One should consult the JDBC driver documentation to determine which connection option is supported.

```
public void setClass_(java.lang.String val)
```

Returns

None.

hasClass

Returns True if the java class has been set.

```
public boolean hasClass_()
```

Returns

True.

omitClass

Sets the java class name to null.

```
public void omitClass_()
```

Returns

None.

getConnectionMethod

Retrieves the connection method.

```
public java.lang.String getConnectionMethod()
```

Returns

Connection method.

setConnectionMethod

Specifies which method is used to connect to the database server.

URL - a JDBC URL provides a way of identifying a data source so that the appropriate driver will recognize it and establish a connection with it. It will use the information in "jdbc url" to establish connection. If URL is used, one does not need to specify the "data source attributes" parameter.

Pooled Data Source - a ConnectionPoolDataSource object for creating PooledConnection objects. A PooledConnection object represents a physical connection and is cached in memory for reuse, which saves the overhead of establishing a new connection. It will use the information in "data source attributes" to establish connection. If this is specified, one does not need to specify the "jdbc url" parameter.

XA Data Source - an XADataSource object for creating XAConnection objects, connections that can be used for distributed transactions. It will use the information in "data source attributes" to establish connection. If this is specified, one does not need to specify the "jdbc url" parameter.

One should make sure that the class specified in "class" parameter supports the connection method that used here.

The default is URL, which is used by Sun's JDBC/ODBC bridge driver.

```
public void setConnectionMethod(java.lang.String val)
```

Returns

None.

hasConnectionMethod

Returns True if the connection method has been set.

```
public boolean hasConnectionMethod()
```

Returns

True.

omitConnectionMethod

Sets the connection method to null.

```
public void omitConnectionMethod()
```

Returns

None.

getJdbcUrl

Retrieves the JDBC URL

```
public java.lang.String getJdbcUrl()
```

Returns

JDBC URL

setJdbcUrl

This is the JDBC URL necessary to gain access to the database.

The URL usually starts with jdbc:, follows by and ends with information for identifying the data source. For JDBC/ODBC bridge, the subprotocol is odbc:. The information that identifies the rest of the data source are usually in {;=].

Consult the documentation of the driver that you use for further detail.

This parameter will be ignored if URL is not selected in the "connection method" section.

For Sun JDBC/ODBC bridge, the URL looks like this: jdbc:odbc:[;=] e.g. jdbc:odbc:myDataSource; Cachesize=300.

```
public void setJdbcUrl(java.lang.String val)
```

Returns

None.

hasJdbcUrl

Returns True if the JDBC URL has been set.

```
public boolean hasJdbcUrl()
```

Returns

True.

omitJdbcUrl

Sets the JDBC URL to null.

```
public void omitJdbcUrl()
```

Returns

None.

getDataSourceAttributeValuePairSeparator

Retrieves the data source attribute separator.

```
public java.lang.String getDataSourceAttributeValuePairSeparator()
```

Returns

Data source attribute separator.

setDataSourceAttributeValuePairSeparator

This entry specifies the character separator used to separate the attribute-value pair used in the "data source attributes" section.

For example, the attribute-value pair "ServerName!myHost" has\"!\" as a separator.

One should select a separator that will NOT be part of the attribute-name of the attribute-value.

The default value is "!".

```
public void setDataSourceAttributeValuePairSeparator(java.lang.String
val)
```

Returns

None.

hasDataSourceAttributeValuePairSeparator

Returns True if the data source Attribute ValuePair Separator has been set.

```
public boolean hasDataSourceATtributeValuePairSeparator()
```

Returns

True.

omitDataSourceAttributeValuePairSeparator

Sets data source Attribute ValuePair Separator to null.

```
public void omitDataSourceAttributeValuePairSeparator()
```

Returns

None.

getDataSourceAttributes

Get the list of data source attributes.

```
public java.lang.String[] getDataSourceAttributes()
```

Returns

java.lang.String[]

setDataSourceAttributes

A list of separated attribute-value pairs.

This information is used to identify the database and set the connection properties. The attribute name should be exactly the same as the one that is specified in the driver documentation and the value should be a valid one. The whole list will be used to specify the connection properties. To disable an attribute, simply un check it.

It will not be used if the "connection method" section is specified as "URL". For example PortNumber!8888.

The separator used in this parameter should match the one specified in the "data source attribute value pair separator" section. By default, the separator used is "!".

```
public void setDataSourceAttributes(java.lang.String[] val)
```

Returns

None.

getDataSourceAttributes

Retrieves the data source attribute of the input index.

```
public java.lang.String getDataSourceAttributes(int i)
```

Returns

Data Source attribute.

setDataSourceAttributes

Sets the data source attributes of the specified index.

```
public void setDataSourceAttributes(int i, java.lang.String val)
```

Returns

None.

countDataSourceAttributes

Returns the number of data source Attributes

```
public int countDataSourceAttributes()
```

Returns

Int.

removeDataSourceAttributes

Removes the data source attribute at specified index.

```
public void removeDataSourceAttributes(int i)
```

Returns

None.

addDataSourceAttributes

Appends the data source attribute at the end.

```
public void addDataSourceAttributes(int i, java.lang.String val)
```

Returns

None

addDataSourceAttributes

Add a data source attribute to a specified index.

```
public void addDataSourceAttributes(java.lang.String val)
```

Returns

None.

clearDataSourceAttributes

Removes all data source attributes.

```
public void clearDataSourceAttributes()
```

Returns

None.

getUserName

Retrieves the user name that the e*Way uses to connect to the database.

```
public java.lang.String getUserName()
```

Returns

User name.

setUserName

Sets the user name that the e*Way uses to connect to the database.

```
public void setUserName(java.lang.String val)
```

Returns

None.

hasUserName

Returns true if the user name has been set.

```
public boolean hasUserName()
```

Returns

True.

omitUserName

Sets the user Name to null.

```
public void omitUserName()
```

Returns

None.

getPassword

Retrieves the password that the e*Way uses to connect to the database.

```
public java.lang.String getPassword()
```

Returns

Password.

setPassword

Sets the internally encrypted password that the e*Way uses to connect to the database.

```
public void setPassword(java.lang.String val)
```

Returns

None.

setPassword_AsIs

Sets the non-encrypted password that the e*Way uses to connect to the database.

```
public void setPassword_AsIs(java.lang.String val)
```

Returns

None.

hasPassword

Returns True if the password has been set.

```
public boolean hasPassword()
```

Returns

True.

omitPassword

Sets the password to null.

```
public void omitPassword()
```

Returns

None.

getTimeout

Retrieves the login timeout in seconds.

```
public java.lang.String getTimeout()
```

Returns

java.lang.String

setTimeout

Sets the login timeout in seconds.

```
public void setTimeout(java.lang.String val)
```

Returns

None.

hasTimeout

Returns True if the time out has been set.

```
public boolean hasTimeout()
```

Returns

True.

omitTimeout

Sets the time out to null.

```
public void omitTimeout()
```

Returns

None.

5.10 Com_stc_jdbcx_dbcfg

```
public class Com_stc_jdbcx_dbcfg
```

Methods of the dbdfg

getDataSource

Returns the Datasource object.

```
public Com_stc_jdbcx_oraclecfg.DataSource getDataSource()
```

Returns

DataSource object.

setDataSource

Sets the DataSource object.

```
public void setDataSource(Com_stc_jdbcx_oraclecfg.DataSource val)
```

Returns

None.

5.11 Monk ODBC e*Way Functions

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

Note: *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.*

This Section Explains:

5.12 Basic Functions

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

The basic functions are:

stop-schedule on page 229

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 will return **#t**, otherwise **#f**.

Parameters

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

Return Values

Boolean

Returns **#t** when successful and **#f** 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.

get-logical-name

Syntax

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

Description

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

Parameters

None.

Return Values

string
Returns the name of the e*Way (as defined by the Schema Designer).

Throws

None.

send-external-down

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown-request 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 66**). 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.

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 will be 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.

5.13 Standard e*Way Functions

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

The standard functions are:

db-stdver-conn-estab

Syntax

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

Description

db-stdver-conn-estab is used to establish external system connection.The following tasks are performed by this function:

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

Parameters

None.

Return Values

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

Throws

None.

Additional Information

In order to use the standard database time format, the following function call has been added to this function (immediately before the call to the **db-bind** function):

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

To override the use of the standard database time format, the **db-std-timestamp-format** function call should be removed.

For "Maximum Long Data Size" the ODBC library allocates an internal buffer for each SQL_LONGVARCHAR and SQL_LONGVARBINARY data, when the SQL statement or stored procedure that contains these data types 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 257 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-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 will be placed in the down state.

Parameters

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

Return Values

A string

A return of "SUCCESS" indicates that the external is ready to suspend.

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 will be 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-conn-ver

Syntax

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

Description

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

Parameters

None.

Return Values

A 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.

This SQL statement is designed for DBMSs other than Oracle; the use of this function occasionally results in an error in the e*Way's log file. Despite the error, the function will complete successfully.

Note: *To users of earlier versions of DART:* ***db-check-connect*** *calls should be replaced with* ***db-alive*** *calls.*

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-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

A string

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

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

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect. Upon reconnecting, this function will be 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

A string

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

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

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect. Upon reconnecting, this function will be 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
  )
))
```

db-stdver-init

Syntax

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

Description

db-stdver-init begins the initialization process for the e*Way. The function loads all of the monk extension library files that the other e*Way functions will access.

Parameters

None.

Return Values

A string

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

Throws

None.

db-stdver-neg-ack

Syntax

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

Description

db-stdver-neg-ack is used to send a negative acknowledgement 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

A 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 will be re-executed.

Throws

None.

Examples

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

db-stdver-pos-ack

Syntax

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

Description

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

Parameters

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

Return Values

A string

An empty string indicates a successful operation. The e*Way will then be 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 will be 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-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

A string

An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way will compare 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:

1. If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
2. 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 will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll 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 will pause 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 47) the message (Event) will be journaled.

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

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

A string

An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way will compare 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:

1. If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
2. 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 will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll 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 will pause 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 47) the message (Event) will be journaled.

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

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-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 will pass the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

A string

SUCCESS allows an immediate shutdown to occur, anything else delays shutdown until (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-startup

Syntax

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

Description

db-stdver-startup is used for instance specific function loads and invokes setup.

Parameters

None.

Return Values

A 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
  )
))
```

5.14 General Connection Functions

The functions in this category control the e*Way's database connection operations.

The general connection functions are:

connection-handle?

Syntax

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

Description

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

Parameters

This function requires a single variable of any datatype.

Return Values

Boolean

Returns **#t** (true) if the argument is a connection handle; otherwise, returns **#f** (false). Use **db-get-error-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-alive

Syntax

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

Description

db-alive is used to determine if the cause of a failing ODBC operation is due to a broken connection. It returns whether or not the database connection was alive during the last call to any ODBC 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) if the connection to the database server is still alive; otherwise, returns **#f** (false) if the connection to the database server is either dead or down. Use **db-get-error-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 will exit the loop only if it loses the connection to the database.

Notes

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

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

3 Once the ODBC e*Way determines 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-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) if successful; otherwise, returns **#f** (false). Use **db-get-error-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 will commit the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

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 the function returns a **#f** value.

Parameters

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

Return Values

A 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.

```
ODBC - odbcmsg.ssc, odbcmsg_display.monk
```

Throws

None.

Examples

Scenario #1 — sample code for db-get-error-str

```
...
(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 will commit the transaction. Otherwise, the application will print out the error message and roll 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)
         (odbcmsg-display (db-get-error-str hdbc))
      )
      (if (not (db-logout hdbc))
         (odbcmsg-display (db-get-error-str hdbc))
      )
```

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

Program output of the above example:

Output of (db-get-error-str hdbc)

```
ODBC|S0002|942|INTERSOLV|ODBC Oracle driver|Oracle|ORA-00942: table
or view does not exist
DART|63|STCDB_X_conn_sql_exec_len||unable to execute SQL statement
```

Output of (odbcmsg-display (db-get-error-str hdbc))

```
ODBC message #0:
msg_source      : ODBC
sql_state       : S0002
native_code     : 942
drv_vendor      : INTERSOLV
component       : ODBC Oracle driver
err_source      : Oracle
msg_string      : ORA-00942: table or view does not exist

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

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.

This function requires an encrypted password. If you have specified a password in the Database Setup section of the e*Way Editor, it has already been encrypted. (See **"Database Setup" on page 66**.)

If you define the password 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 standard encrypt-password function 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** (true) if the argument is a connection handle; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

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

Explanation

The above example shows how to use the connection handle (hdbc) to log into the data source “Payroll” as “James”.

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) )
(define uid "James")
(define pwd)
(if (db-login hdbc "Payroll" "James" "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

Make sure you roll back or commit a transaction before you call **db-logout**. If a transaction is neither committed nor rolled back, it will be automatically rolled back before logout.

db-max-long-data-size

Syntax

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

Description

db-max-long-data-size specifies the maximum buffer size for the long data.(SQL_LONGVARCHAR, SQL_LONGVARBINARY) Long data may have a range in size up to 2 gigabytes ($2x10^9$). In order to limit the memory consumption of the ODBC library, it is necessary to use this function to specify the maximum data size expected. Long data larger than the specified size will be truncated. This data size will be 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 data type. Note: The default buffer size is 1 megabyte.

Return Values

Boolean

Returns #t (true) if successful; and 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 1 megabyte (1048576). It is not necessary to call this function unless the long data is in excess of 1 megabyte.

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) if successful; otherwise, returns **#f** (false). Use **db-get-error-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 will commit the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

make-connection-handle

Syntax

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

Description

make-connection-handle constructs the *connection handle*.

Parameters

None.

Return Values

A handle
Returns a connection-handle if successful, otherwise;

Boolean
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

```
(let ((hConnection (make-connection-handle)))
    (if (connection-handle? hConnection)
        (begin
            (display "Established a valid connection handle\n")
        )
        (begin
            (display "Failed to get a connection handle: ")
            (display (db-get-error-str connection-handle))
            (newline)
        )
    )
)
```

Explanation

The above example creates a connection handle variable called hConnection. The results are verified by using the **connection-handle?** function to check the type of the hConnection variable. If the results are a connection handle, then the message "Established a valid connection handle" is displayed. If the return value is not a connection handle, then the message "Failed to get a connection handle:" and the error string are displayed.

statement-handle?

Syntax

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

Description

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

Parameters

This function requires a single variable of any datatype.

Return Values

Boolean

Returns **#t** (true) if the argument is a statement handle; otherwise, returns **#f** (false). Use **db-get-error-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.

5.15 Static SQL Functions

The functions in this category control the e*Way's interaction with static SQL commands.

The static SQL functions are:

Static vs. 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. Dynamic statements do not require knowledge of the complete structure on 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 80 on page 265** shows the difference between Dynamic SQL with immediate execution and Dynamic SQL with prepared execution.

Benefits of Dynamic SQL

Using dynamic SQL commands, an application can prepare a "generic" SQL statement once and execute it multiple times. 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

The use of dynamic SQL commands has some significant limitations. 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.

Figure 77 Calling a Stored Procedure (Oracle)

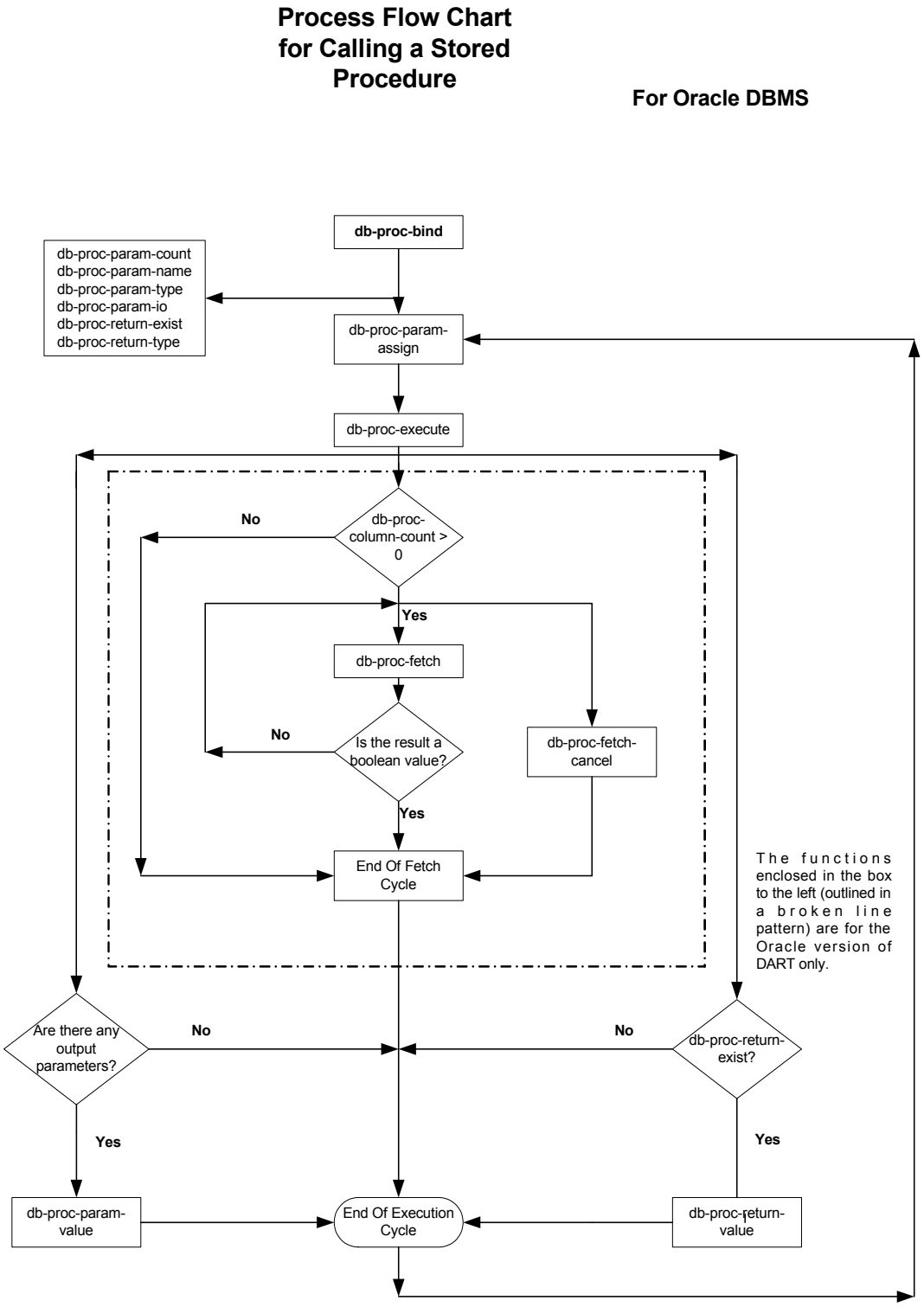


Figure 78 Calling a Stored Procedure (Sybase)

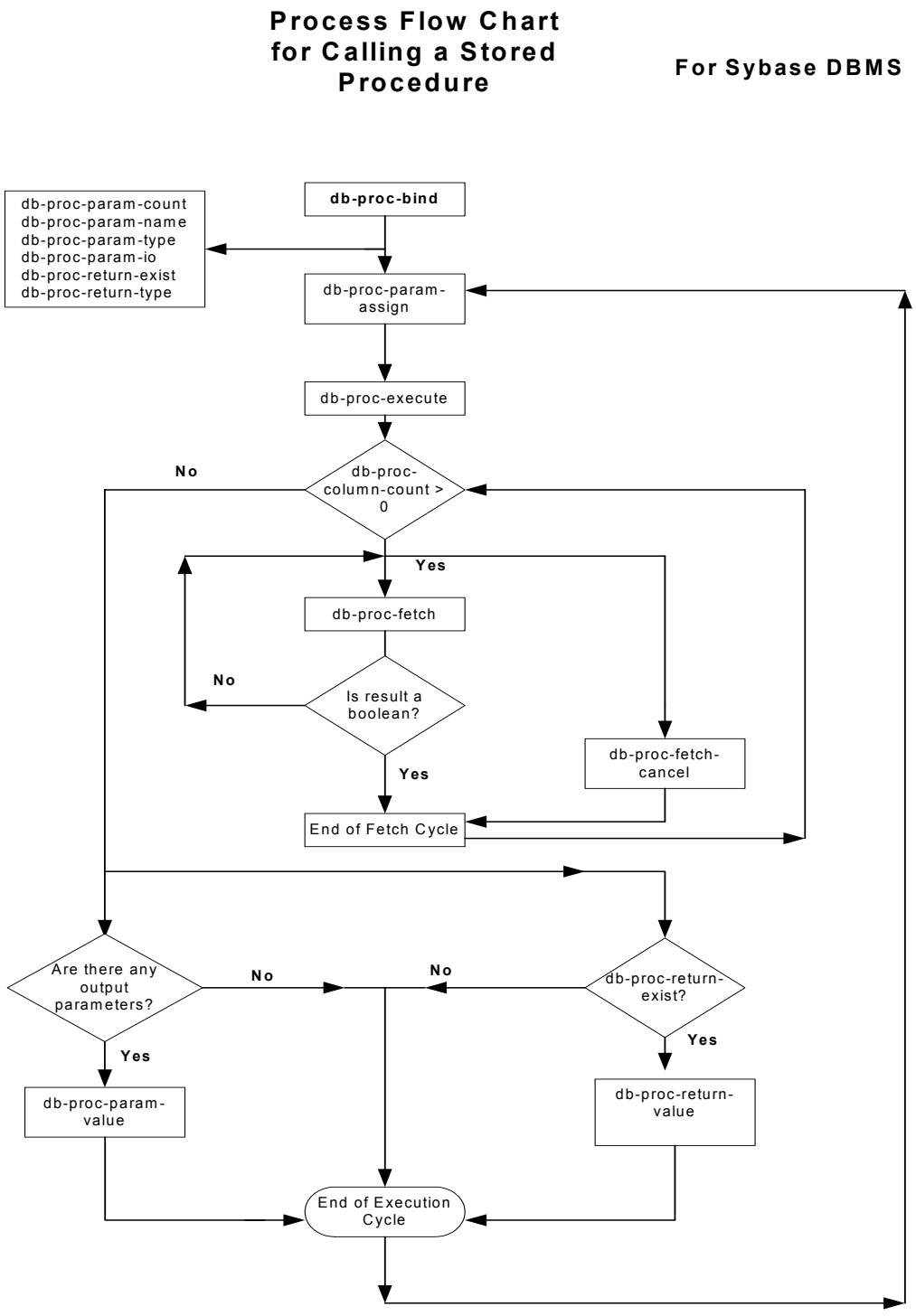


Figure 79 Dynamic Statement Flow Chart

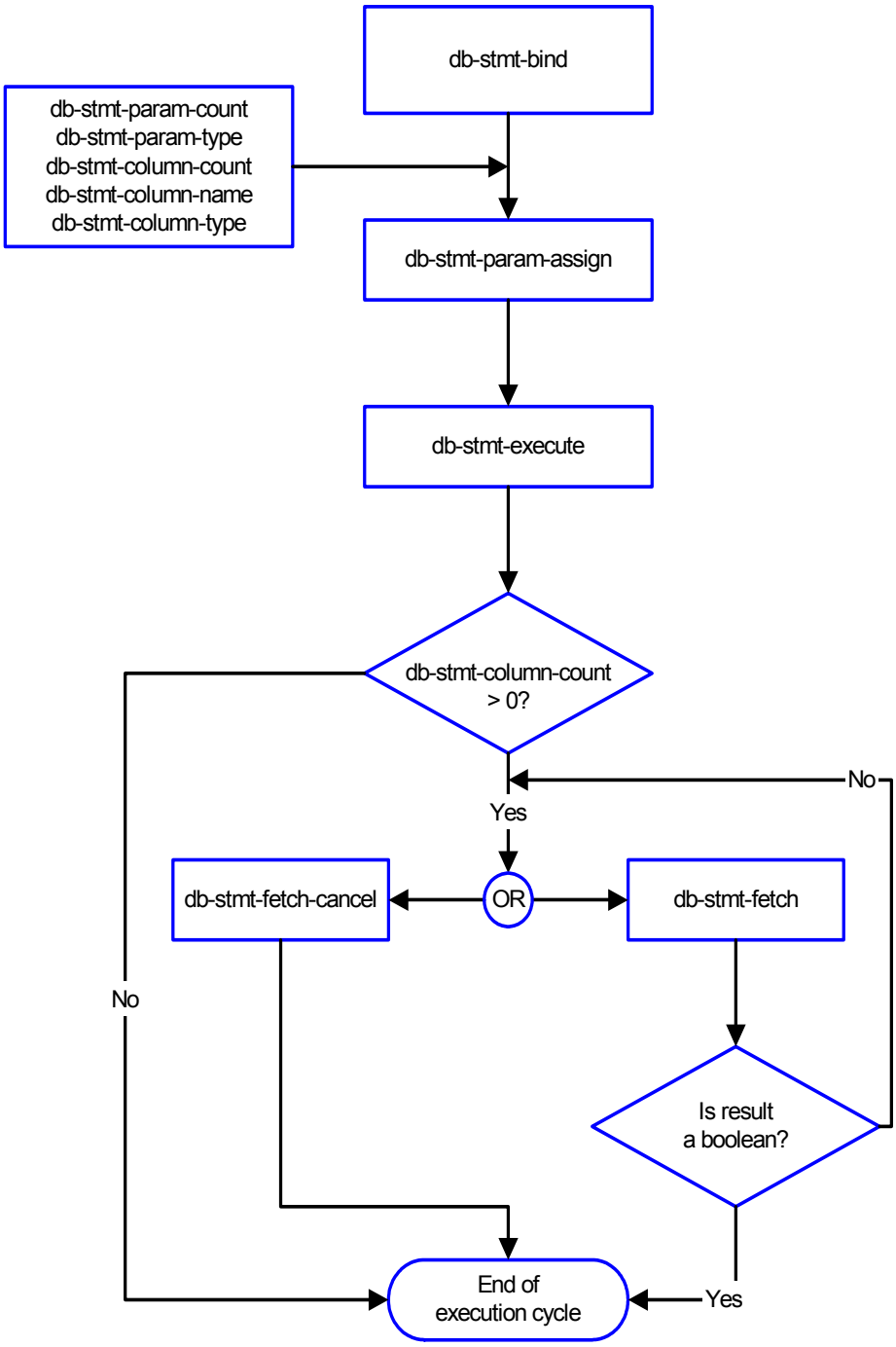
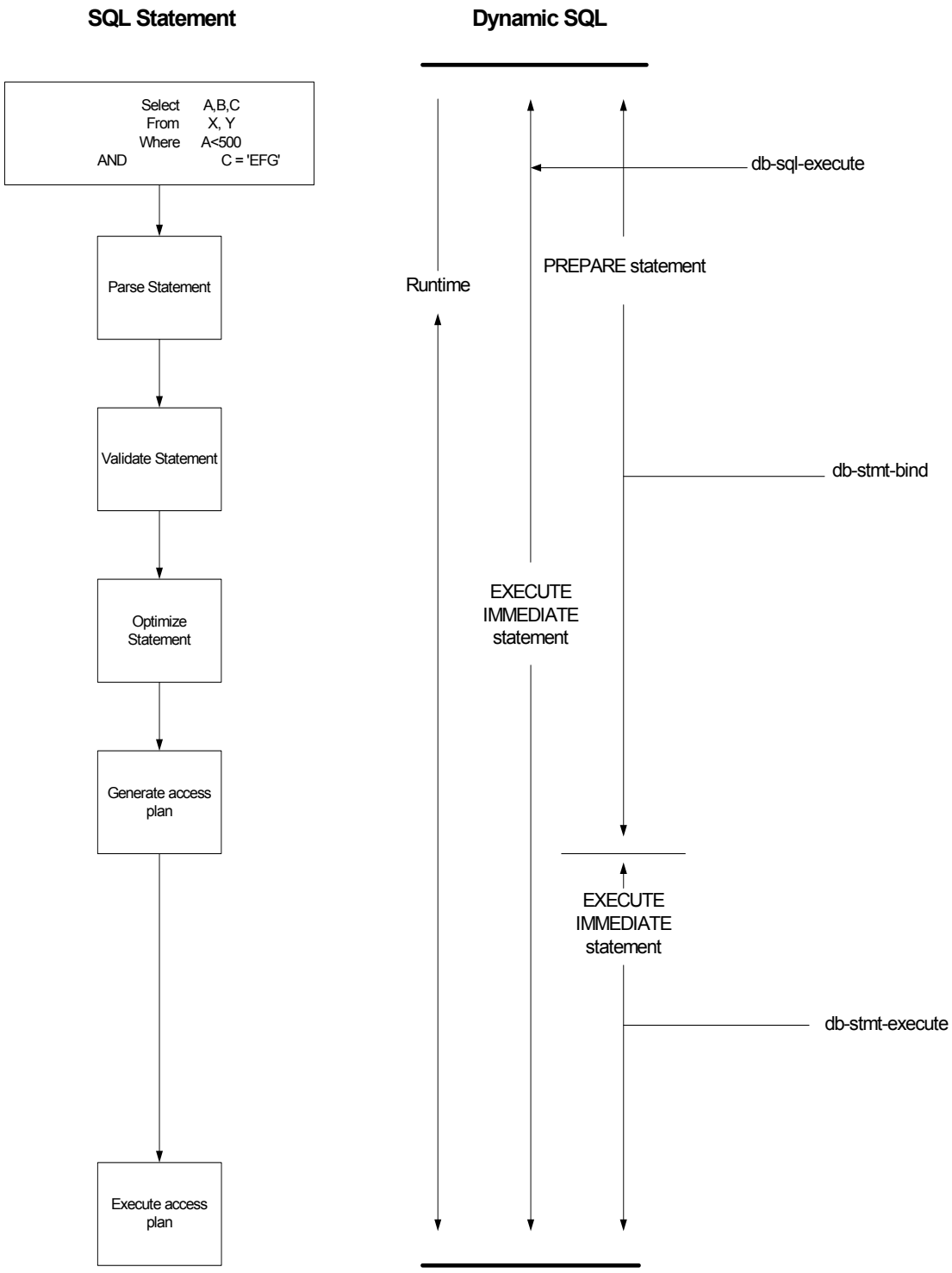


Figure 80 Example of Dynamic SQL processing



ODBC SQL Type Support

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

Table 17 ODBC SQL Type Support

SQL Type Name	SQL Datatype	Oracle Datatype
SQL_BIT	BIT	N/A
SQL_BINARY	BINARY (n)	N/A
SQL_VARBINARY	VARBINARY (n)	RAW (n)
SQL_CHAR	CHAR (n)	CHAR (n)
SQL_VARCHAR	VARCHAR (n)	VARCHAR2 (n)
SQL_DECIMAL	DECIMAL (p, s)	NUMBER (p, s)
SQL_NUMERIC	NUMERIC (p, s)	N/A
SQL_TINYINT	TINYINT	+
SQL_BIGINT	BIGINT	+
SQL_SMALLINT	SMALLINT	+
SQL_INTEGER	INTEGER	+
SQL_REAL	REAL	*
SQL_FLOAT	FLOAT(p)	FLOAT(b)
SQL_DOUBLE	DOUBLE PRECISION	FLOAT
SQL_DATE	DATE	N/A
SQL_TIME	TIME	N/A
SQL_TIMESTAMP	TIMESTAMP	DATE
SQL_LONGVARCHAR	LONG VARCHAR	LONG
SQL_LONGVARBINARY	LONG VARBINARY	LONG RAW

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

+Oracle uses number (p) to define datatypes that span TININT, BIGINT, SMALLINT, and INTEGER. Oracle int type is internally mapped to NUMBER (38) which will be returned as SQL_DECIMAL.

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

SQL_DECIMAL and SQL_NUMERIC datatypes require specification of scale which indicates the number of digits to the right of the decimal point.

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

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

Boolean
If the selection-name 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-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 will display 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.

A 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**. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title= 'manager'")
        (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 will display 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

A 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 0)) ((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)
  )
```

```
)
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program will loop 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

1. 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.
2. If the value of the column contains the comma (,) character, the user will be 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.

db-sql-execute

Syntax

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

Description

db-sql-execute executes 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) if successful; otherwise, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-login hdbc "Payroll" "James" "pwd")
   (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 will insert a record into the table "employee."

Notes

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

The **db-sq.-execute** function can no longer be used to commit and roll back transactions. Instead, use **db-commit** or **db-rollback**.

Note: *The Merant ODBC drivers limit the size of the SQL statement to 32 Kbyte.*

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

A 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-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 will be formatted. If the *data-string* is an empty string, the procedure will return a NULL value for all SQL datatypes 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	An SQL datatype string, i.e., SQL_VARCHAR.

Return Values

A 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.

Notes

1. For SQL_CHAR and SQL_VARCHAR (SQL datatypes) *db-sql-format* will place 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.
2. 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"
   ```

   For SQL_CHAR and SQL_VARCHAR (SQL datatypes) **db-sql-format** will place 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.

   The following table shows the typical *data-string* and the corresponding results of the formatting for the OBDC e*Way.

Table 18 SQL Statement Format

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_DATE	1998-02-19	{d ‘1998-02019’}
SQL_DATE	19980219	{d ‘1998-02-19’}
SQL_TIME	12 :34:56	{t ‘12:34:56’}
SQL_TIME	1234	{t ‘12:34:00’}
SQL_TIMESTAMP	1998-02-19 12:34:56.789	{ts ‘1998-02-19 12:34:56.789’}

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) 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
        (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.

Note: *The Merant ODBC drivers limit the size of the SQL statement to 32 Kbyte.*

5.16 Dynamic SQL Functions

The function sin this category control the e*Way's interaction with dynamic SQL commands. For information about the differences between static and dynamic SQL functions, see **"Static vs. Dynamic SQL Functions" on page 261**.

The dynamic SQL functions are:

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 should be input or output parameters with 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** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Additional Information

If the user needs to input /output binary data in the raw (binary) format, they should use db-stmt-bind-binary.

Notes

1. Oracle OCI API is unable to report the datatype for each bound parameter in a dynamic statement. All bound parameters will default to VARCHAR datatypes. This will allow Oracle to implicitly convert the data string of each parameter into the correct data value of the parameter at the execution of the dynamic statement.
2. If the user needs to select the long datatype column, the long column should appear at the end of the selection list.

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 will be 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-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	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

A number

Returns a number greater than zero (0) when the record set is available.

Boolean

If no record set is available, the return value will be **#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	statement handle	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

A 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 returns 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 dynamic statement specified.
index	integer	An integer equal to -- 0 to db-stmt-column-count minus 1.

Return Values

A 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-execute

Syntax

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

Description

db-stmt-execute executes the dynamic statement of a specified statement-handle.

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-stmt-bind.

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	string	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 no more records are available to "fetch."

Boolean

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	string	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-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-stmt-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-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-stmt-bind.

Return Values

An Integer

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

Boolean

If unsuccessful, 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-stmt-bind.
index	integer	The number between 0 and db-stmt-param-count minus 1.

Return Values

A 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.

db-stmt-row-count

Syntax

```
(db-stmt-row-count connection-handle statement-handle index)
```

Description

db-stmt-row-count returns 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	statement handle	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

Boolean

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

Throws

None.

5.17 Stored Procedure Functions

The functions in this category control the e*Way's interaction with stored procedures.

The stored procedure functions are:

Benefits of Stored Procedures

When a 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 procedure.

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 a proc-handle if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hstmt (db-proc-bind hdbc "test")
(if (not (statement-handle? hstmt)
    (display "fail to bind stored procedure test\n")
)
```

Notes

ODBC does not recognize any procedure inside the PACKAGE of the Oracle DBMS. Therefore, you cannot bind any procedure defined inside the PACKAGE of the Oracle DBMS.

The procedure name is limited to 30 characters.

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 format of the input and output data is binary.

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

A string
Returns a statement-handle when successful; otherwise

Boolean
Returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

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

A number

Returns a number greater than zero (0) when the record set is available.

Boolean

If no record set is available, the return value will be **#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)
)
```

Notes

1 The Oracle procedure to return the result set is given here:

```
oracle_odbc.sql
```

2 The ODBC configuration parameter to set up the return result set must show

```
ProcedureRetResult = 1
```

For more information see **"Sample .odbc.ini File" on page 29**.

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

A 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)
    )
```

Notes

Since the result set of a stored procedure is returned through the parameters of the PL/SQL table type, the name of the table type parameter will be returned.

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	string	SQL datatype of the specified column in the record set --0 to db-proc-column-count minus 1.

Return Values

A 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)
    )
```

Notes

Since the result set of the stored procedure is returned through the parameters of the PL/SQL table type, a PL/SQL table can only contain one standard Oracle datatype.

db-proc-execute

Syntax

```
(db-proc-execute  connection-handle  statement-handle)
```

Description

db-proc-execute executes out 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 will fail if the exponential part of the value is larger than 38.

For example:

- 1.555E+38 is acceptable
- 1.55E+39 will prevent 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

A 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

```
(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)
    )
```

Notes

1 The Oracle procedure to return result set is given here:

```
oracle_odbc.sql
```

2 ODBC configuration parameter to set up the capability of return result set must show

```
ProcedureRetResult = 1
```

For more information see Sample**.odbc.ini** **"Sample .odbc.ini File" on page 29**.

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

```
(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)
)
```

db-proc-param-assign

Syntax

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

Description

db-proc-param-assign "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 dynamic statement 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 Scenario #2 on above.

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

A 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))
)
```

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 dynamic statement 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

A 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-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.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

A 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 dynamic statement 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

A 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-value

Syntax

```
(db-proc-param-value connection-handle statement-handle param-index)
```

Description

db-proc-param-value is used to retrieve 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 dynamic statement 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

A 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)
)
```

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 dynamic statement 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))
                )
          ...
          ...
    )
```

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

A 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)
  )
```

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

A 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

1. 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. The SQL Server has a defined set of return values; or users can define their own return values.
2. The SQL Server reserves 0 to indicate a successful return, and negative values in the range of -1 to -99 are assigned to a listing of reasons for failure. Numbers 0 and -1 to -14 are in use currently (see below).

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

5.18 Message Event Functions

The functions in this category control the e*Way's message Event operations.

The message Event functions are:

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 DART Event Type Definition, retrieves all procedure 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.
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-error-str** to retrieve the error message.

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-stmt-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-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 an 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 in 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-error-str** to retrieve 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-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 SQLSELECT 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.

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 in 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-error-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 Event Definition 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-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 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 in 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-error-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)
             ($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 Collaboration Rules file 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, the Collaboration procedure 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.

Notes

1. 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**.
2. 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.

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 in 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; otherwise, returns **#f** (false). Use **db-get-error-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].

5.19 Sample Monk Scripts

This section includes sample Monk scripts which demonstrate how to use the ODBC e*Way's Monk functions. These Monk scripts demonstrate the following activities:

5.19.1 Initializing Monk Extensions

The sample script shows how to initialize the Monk extensions. This function is used by many of the other sample Monk scripts shown in this chapter.

To use this sample script in an actual implementation, modify the following values:

- **EGATE** – This designates the location of the e*Gate client.
- **dsn** – This is he name of the data source.
- **uid** – This is the user name.
- **pwd** – This is the login password.

```
;demo-init.monk

(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 STCDB variables, data source, user ID, and password
;;

(define STCDB "ORACLE8")

(load-library "stc_dbmonkext.dll")

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

5.19.2 Calling Stored Procedures

This script gives an example of calling Stored Procedures. See **"Stored Procedure Functions" on page 291** for more details.

```
;demo-proc-execute.monk

; 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 "\ndatabase login succeed !\n")

      ; bind the stored procedure
      (define hstmt1 (bind-procedure hdbc "PERSONNEL.GET_EMPLOYEES"))

      ; call the stored procedure if the binding is successful
      (if (statement-handle? hstmt1)
         (begin
            (display "call PERSONNEL.GET_EMPLOYEES to get all sales
...\n\n")
            (if (and
                  (db-proc-param-assign hdbc hstmt1 0 "30")
                  (db-proc-param-assign hdbc hstmt1 1 "10")
                )
               (execute-procedure hdbc hstmt1)
               (display (db-get-error-str hdbc))
            )
         )
      )
```

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

5.19.3 Inserting Records with Dynamic SQL Statements

```
;demo-stmt-insert.monk

; 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 SCOTT.BONUS SELECT ENAME, JOB, SAL, COMM
FROM SCOTT.EMP WHERE DEPTNO = ?")

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

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
           (display "\nInsert accounting department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "10")
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the insertions ...\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 "\nInsert sales department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "20")
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the insertions ...\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))
         )
```

5.19.4 Updating Records with Dynamic SQL Statements

```
;demo-stmt-update.monk

; 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 SCOTT.BONUS SET COMM = ? WHERE JOB = ?")

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

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nUpdate commission of manager ...\n")
            (if
               (and
                  (db-stmt-param-assign hdbc hstmt1 0 "10")
                  (db-stmt-param-assign hdbc hstmt1 1 "MANAGER")
               )
               (if (execute-statement hdbc hstmt1)
                  (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 "\nUpdate commission of clerk ...\n")
            (if
               (and
                  (db-stmt-param-assign hdbc hstmt1 0 "20")
                  (db-stmt-param-assign hdbc hstmt1 1 "CLERK")
               )
               (if (execute-statement hdbc hstmt1)
                  (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))
                     )
                  )
               )

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

5.19.5 Selecting Records with Dynamic SQL Statements

```
;demo-stmt-select.monk

; 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 EMPNO, ENAME, JOB FROM SCOTT.EMP WHERE JOB = ?")
(define stmt2 "SELECT ENAME, DNAME, JOB, HIREDATE FROM SCOTT.EMP,
SCOTT.DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO AND DEPT.DNAME = ?")

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

      ; 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 all salesman ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "SALESMAN")
               (if (not (execute-statement hdbc hstmt1))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
            (display "\nList all manager ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "MANAGER")
               (if (not (execute-statement hdbc hstmt1))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
         )
      )

      (if (statement-handle? hstmt2)
         (begin
            (display "\nList employee of accounting department
...\n\n")
            (if (db-stmt-param-assign hdbc hstmt2 0 "ACCOUNTING")
               (if (not (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))
         )
```

5.19.6 Deleting Records with Dynamic SQL Statements

```
;demo-stmt-delete.monk

; 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 SCOTT.BONUS WHERE ENAME IS NOT NULL")

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

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nDelete records from scott.bonus table ...\n")
            (if (execute-statement hdbc hstmt1)
               (begin
                  (display "\nCommit the deletions ...\n")
                  (if (not (db-commit 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))
)
```

5.19.7 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 **"Static SQL Functions" on page 261** and **"Dynamic SQL Functions" on page 278** for more details.

```
;demo-image-insert.monk

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

(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 "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 (display-stmt-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 (bind-image-statement hdbc stmt)
   (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
      (display (string-append "\nDynamic statement : " stmt "\n"))
      (if (statement-handle? hstmt)
         (begin
;            (db-stmt-param-bind hdbc hstmt 0 "SQL_INTEGER" 4 0)
;            (db-stmt-param-bind hdbc hstmt 1 "SQL_VARCHAR" 20 0)
;            (db-stmt-param-bind hdbc hstmt 2 "SQL_VARCHAR" 10 0)
;            (db-stmt-param-bind hdbc hstmt 3 "SQL_INTEGER" 38 0)
;            (db-stmt-param-bind hdbc hstmt 4 "SQL_INTEGER" 38 0)
;            (db-stmt-param-bind hdbc hstmt 5 "SQL_INTEGER" 10 0)
            (db-stmt-param-bind hdbc hstmt 6 "SQL_LONGVARBINARY"
2000000 0)
            (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
   )
)


(define image1-id "7100")
(define image1-name "Coast")
(define image1-type "JPEG")
(define image1-width "1280")
(define image1-height "1024")
(define image1-file (string-append image1-name ".jpg"))

(define image-port (open-input-file image1-file))
(define image1-data (read image-port 1000000))
(close-port image-port)
(define image1-size (number->string (string-length image1-data)))

(define image2-id "7200")
(define image2-name "Launch")
(define image2-type "JPEG")
(define image2-width "2000")
(define image2-height "1600")
(define image2-file (string-append image2-name ".jpg"))

(define image-port (open-input-file image2-file))
(define image2-data (read image-port 2000000))
(close-port image-port)
(define image2-size (number->string (string-length image2-data)))

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

(define stmt0 "select count(0) from SCOTT.IMAGE where PIX_ID = ?")
(define stmt1 "insert into SCOTT.IMAGE (PIX_ID, PIX_NAME, PIX_TYPE,
BYTE_SIZE, PIX_WIDTH, PIX_HEIGHT, PIX_DATA) values (?, ?, ?, ?, ?, ?,
?)")

(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-max-long-data-size hdbc 2000000)) (newline)

    ; bind the query and insert statement
    (define hquery  (bind-statement hdbc stmt0))
    (define hinsert (bind-image-statement hdbc stmt1))

    (if (and
       (statement-handle? hquery)
       (statement-handle? hinsert)
     )
     (begin
       (if (not (query-exist hdbc hquery image1-id))
         (begin
           (display (string-append "insert image " image1-file "\n"))
           (if (and
              (db-stmt-param-assign hdbc hinsert 0 image1-id)
              (db-stmt-param-assign hdbc hinsert 1 image1-name)
              (db-stmt-param-assign hdbc hinsert 2 image1-type)
              (db-stmt-param-assign hdbc hinsert 3 image1-size)
              (db-stmt-param-assign hdbc hinsert 4 image1-width)
              (db-stmt-param-assign hdbc hinsert 5 image1-height)
              (db-stmt-param-assign hdbc hinsert 6 image1-data)
            )
            (if (execute-statement hdbc hinsert)
              (db-commit hdbc)
              (display (db-get-error-str hdbc))
            )
            (display (db-get-error-str hdbc))
          )
         )
       )

       (if (not (query-exist hdbc hquery image2-id))
         (begin
           (display (string-append "insert image " image2-file "\n"))
           (if (and
              (db-stmt-param-assign hdbc hinsert 0 image2-id)
              (db-stmt-param-assign hdbc hinsert 1 image2-name)
              (db-stmt-param-assign hdbc hinsert 2 image2-type)
              (db-stmt-param-assign hdbc hinsert 3 image2-size)
              (db-stmt-param-assign hdbc hinsert 4 image2-width)
              (db-stmt-param-assign hdbc hinsert 5 image2-height)
              (db-stmt-param-assign hdbc hinsert 6 image2-data)
            )
            (if (execute-statement hdbc hinsert)
              (db-commit 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))
 )
```

5.19.8 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 **"Static SQL Functions" on page 261** and **"Dynamic SQL Functions" on page 278** for more details.

```
;demo-image-select.monk

; 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 "BITMAP") (set! file_name (string-append
first_name ".bmp")))
         ((string=? file_type "TIFF") (set! file_name (string-append
first_name ".tif")))
         (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 stmt "select PIX_NAME, PIX_TYPE, PIX_WIDTH, PIX_HEIGHT,
PIX_DATA from SCOTT.IMAGE where PIX_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-max-long-data-size hdbc 2000000)) (newline)

      ; bind the select statement
      (define hselect (bind-binary-statement hdbc stmt))

      ; execute the dynamic statement
      (display "select IMAGE table\n")
      (if (statement-handle? hselect)
         (begin
            (if (db-stmt-param-assign hdbc hselect 0 "7100")
               (if (not (execute-statement hdbc hselect))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
            (if (db-stmt-param-assign hdbc hselect 0 "7200")
               (if (not (execute-statement hdbc hselect))
                  (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))
)
```

5.19.9 Common Supporting Routines

This sample script displays and defines values and parameters for stored procedures. The routines contained in this script are used by many of the Monk samples in this chapter. For more details about functions used in this script, see **"Stored Procedure Functions" on page 291**

```
;demo-common.monk

;;
;; 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
   )
)
```

Chapter 6

Monk ODBC e*Way Functions

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

Note: *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.*

This Chapter Explains:

6.1 Basic Functions

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

The basic functions are:

event-send-to-egate

Syntax

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

Description

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

Parameters

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

Return Values

Boolean

Returns **#t** when successful and **#f** 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.

get-logical-name

Syntax

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

Description

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

Parameters

None.

Return Values

string
Returns the name of the e*Way (as defined by the Schema Designer).

Throws

None.

send-external-down

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

send-external-up

Syntax

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

Description

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

Parameters

None.

Return Values

None.

Throws

None.

shutdown-request

Syntax

```
(shutdown-request)
```

Description

shutdown-request 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 66**). 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.

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 will be 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.

6.2 Standard e*Way Functions

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

The standard functions are:

db-stdver-conn-estab

Syntax

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

Description

db-stdver-conn-estab is used to establish external system connection.The following tasks are performed by this function:

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

Parameters

None.

Return Values

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

Throws

None.

Additional Information

In order to use the standard database time format, the following function call has been added to this function (immediately before the call to the **db-bind** function):

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

To override the use of the standard database time format, the **db-std-timestamp-format** function call should be removed.

For "Maximum Long Data Size" the ODBC library allocates an internal buffer for each SQL_LONGVARCHAR and SQL_LONGVARBINARY data, when the SQL statement or stored procedure that contains these data types 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 383 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-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 will be placed in the down state.

Parameters

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

Return Values

A string

A return of "SUCCESS" indicates that the external is ready to suspend.

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 will be 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-conn-ver

Syntax

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

Description

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

Parameters

None.

Return Values

A 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.

This SQL statement is designed for DBMSs other than Oracle; the use of this function occasionally results in an error in the e*Way's log file. Despite the error, the function will complete successfully.

Note: *To users of earlier versions of DART:* ***db-check-connect*** *calls should be replaced with* ***db-alive*** *calls.*

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-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

A string

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

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

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect. Upon reconnecting, this function will be 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

A string

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

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

CONNERR indicates the loss of connection with the external, client moves to a down state and attempts to connect. Upon reconnecting, this function will be 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
  )
))
```

db-stdver-init

Syntax

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

Description

db-stdver-init begins the initialization process for the e*Way. The function loads all of the monk extension library files that the other e*Way functions will access.

Parameters

None.

Return Values

A string

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

Throws

None.

db-stdver-neg-ack

Syntax

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

Description

db-stdver-neg-ack is used to send a negative acknowledgement 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

A 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 will be re-executed.

Throws

None.

Examples

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

db-stdver-pos-ack

Syntax

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

Description

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

Parameters

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

Return Values

A string

An empty string indicates a successful operation. The e*Way will then be 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 will be 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-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

A string

An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way will compare 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:

1. If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
2. 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 will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll 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 will pause 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 47) the message (Event) will be journaled.

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

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

A string

An empty string indicates a successful operation.

RESEND causes the message to be immediately resent. The e*Way will compare 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:

1. If the number of attempts does not exceed the maximum, the e*Way will pause the number of seconds specified by the **Resend Timeout** parameter, increment the "resend attempts" counter for that message, then repeat the attempt to send the message.
2. 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 will pause the number of seconds specified by the **Resend Timeout** parameter. Then, the e*Way will call the **External Connection Establishment function according to the Down Timeout schedule, and** will roll 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 will pause 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 47) the message (Event) will be journaled.

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

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-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 will pass the string "SHUTDOWN_NOTIFICATION" as the parameter.

Return Values

A string

SUCCESS allows an immediate shutdown to occur, anything else delays shutdown until (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-startup

Syntax

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

Description

db-stdver-startup is used for instance specific function loads and invokes setup.

Parameters

None.

Return Values

A 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
  )
))
```

6.3 General Connection Functions

The functions in this category control the e*Way's database connection operations.

The general connection functions are:

connection-handle?

Syntax

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

Description

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

Parameters

This function requires a single variable of any datatype.

Return Values

Boolean

Returns **#t** (true) if the argument is a connection handle; otherwise, returns **#f** (false). Use **db-get-error-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-alive

Syntax

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

Description

db-alive is used to determine if the cause of a failing ODBC operation is due to a broken connection. It returns whether or not the database connection was alive during the last call to any ODBC 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) if the connection to the database server is still alive; otherwise, returns **#f** (false) if the connection to the database server is either dead or down. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define pwd (encrypt-password "uid" "pwd"))
(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 will exit the loop only if it loses the connection to the database.

Notes

1. Most ODBC procedures can detect a dead connection handle except **db-commit** and **db-rollback**. Therefore, when the ODBC procedure returns false, users must check for loss of connection.
2. Once the **db-alive** returns **#f** to indicate either a dead connection handle or an unavailable database server, all the subsequent ODBC function calls associated with that connection handle will not be executed, with the exception of **db-logout**. Each of these procedures will return false with a "lost database connection" error message.
3. Once the ODBC e*Way determines 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-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) if successful; otherwise, returns **#f** (false). Use **db-get-error-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 will commit the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

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 the function returns a **#f** value.

Parameters

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

Return Values

A 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.

```
ODBC - odbcmsg.ssc, odbcmsg_display.monk
```

Throws

None.

Examples

Scenario #1 — sample code for db-get-error-str

```
...
(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 will commit the transaction. Otherwise, the application will print out the error message and roll 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)
         (odbcmsg-display (db-get-error-str hdbc))
      )
      (if (not (db-logout hdbc))
         (odbcmsg-display (db-get-error-str hdbc))
      )
```

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

Program output of the above example:

Output of (db-get-error-str hdbc)

```
ODBC|S0002|942|INTERSOLV|ODBC Oracle driver|Oracle|ORA-00942: table
or view does not exist
DART|63|STCDB_X_conn_sql_exec_len||unable to execute SQL statement
```

Output of (odbcmsg-display (db-get-error-str hdbc))

```
ODBC message #0:
msg_source      : ODBC
sql_state       : S0002
native_code     : 942
drv_vendor      : INTERSOLV
component       : ODBC Oracle driver
err_source      : Oracle
msg_string      : ORA-00942: table or view does not exist

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

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.

This function requires an encrypted password. If you have specified a password in the Database Setup section of the e*Way Editor, it has already been encrypted. (See **"Database Setup" on page 66**.)

If you define the password 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 standard encrypt-password function 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** (true) if the argument is a connection handle; otherwise, returns **#f** (false). Use **db-get-error-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" pwd)
)
```

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) )
(define uid "James")
(define pwd (encrypt-password uid "12345"))
(if (db-login hdbc "Payroll" "James" 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

Make sure you roll back or commit a transaction before you call **db-logout**. If a transaction is neither committed nor rolled back, it will be automatically rolled back before logout.

db-max-long-data-size

Syntax

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

Description

db-max-long-data-size specifies the maximum buffer size for the long data.(SQL_LONGVARCHAR, SQL_LONGVARBINARY) Long data may have a range in size up to 2 gigabytes ($2x10^9$). In order to limit the memory consumption of the ODBC library, it is necessary to use this function to specify the maximum data size expected. Long data larger than the specified size will be truncated. This data size will be 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 data type. Note: The default buffer size is 1 megabyte.

Return Values

Boolean

Returns #t (true) if successful; and 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 1 megabyte (1048576). It is not necessary to call this function unless the long data is in excess of 1 megabyte.

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) if successful; otherwise, returns **#f** (false). Use **db-get-error-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 will commit the transaction specified by the connection. Otherwise, it prints out the error message and rolls back the transaction.

make-connection-handle

Syntax

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

Description

make-connection-handle constructs the *connection handle*.

Parameters

None.

Return Values

A handle
Returns a connection-handle if successful, otherwise;

Boolean
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

```
(let ((hConnection (make-connection-handle)))
    (if (connection-handle? hConnection)
        (begin
            (display "Established a valid connection handle\n")
        )
        (begin
            (display "Failed to get a connection handle: ")
            (display (db-get-error-str connection-handle))
            (newline)
        )
    )
)
```

Explanation

The above example creates a connection handle variable called hConnection. The results are verified by using the **connection-handle?** function to check the type of the hConnection variable. If the results are a connection handle, then the message "Established a valid connection handle" is displayed. If the return value is not a connection handle, then the message "Failed to get a connection handle:" and the error string are displayed.

statement-handle?

Syntax

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

Description

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

Parameters

This function requires a single variable of any datatype.

Return Values

Boolean

Returns **#t** (true) if the argument is a statement handle; otherwise, returns **#f** (false). Use **db-get-error-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.

6.4 Static SQL Functions

The functions in this category control the e*Way's interaction with static SQL commands.

The static SQL functions are:

Static vs. 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. Dynamic statements do not require knowledge of the complete structure on 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 84 on page 391** shows the difference between Dynamic SQL with immediate execution and Dynamic SQL with prepared execution.

Benefits of Dynamic SQL

Using dynamic SQL commands, an application can prepare a "generic" SQL statement once and execute it multiple times. 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

The use of dynamic SQL commands has some significant limitations. 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.

Figure 81 Calling a Stored Procedure (Oracle)

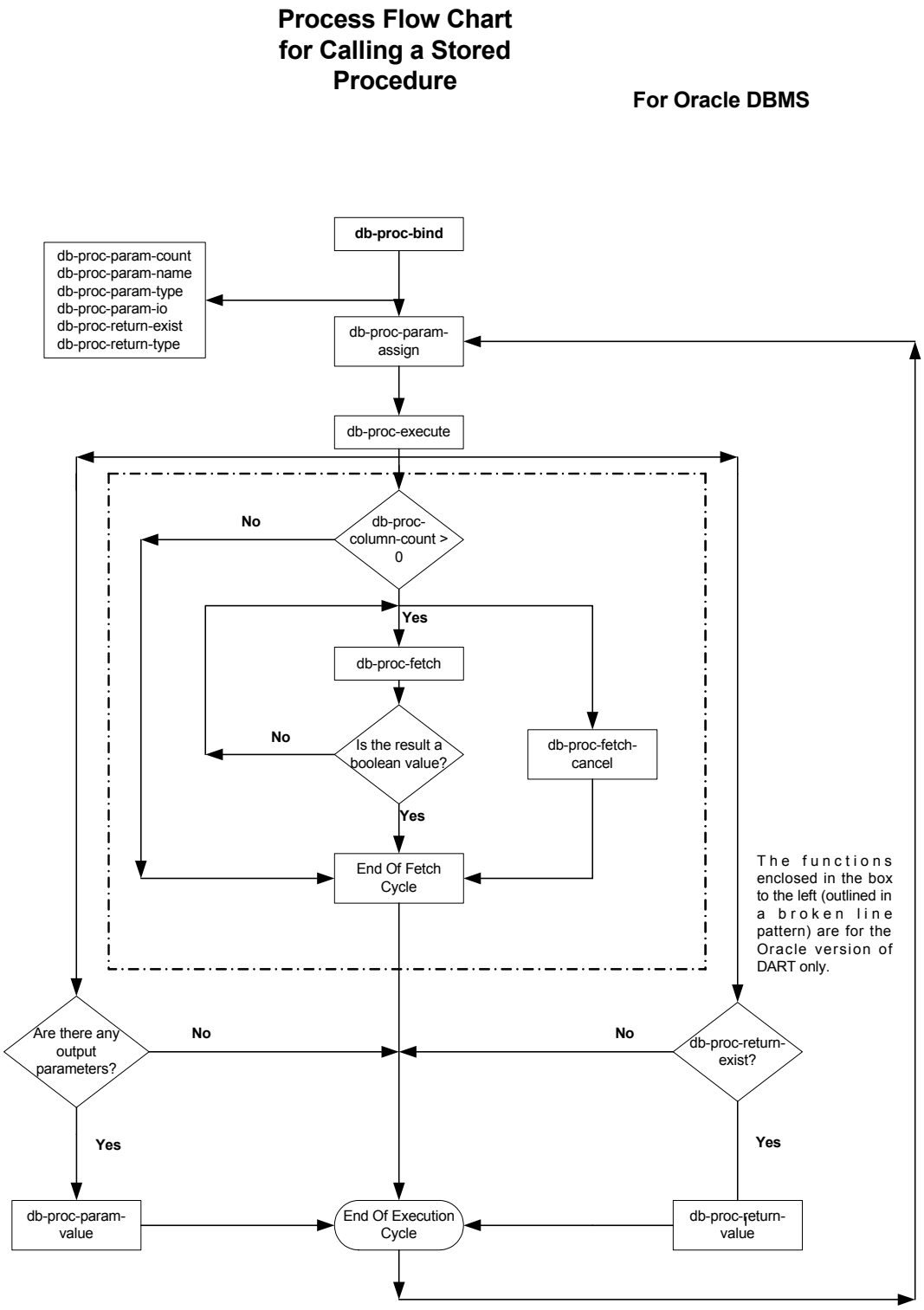


Figure 82 Calling a Stored Procedure (Sybase)

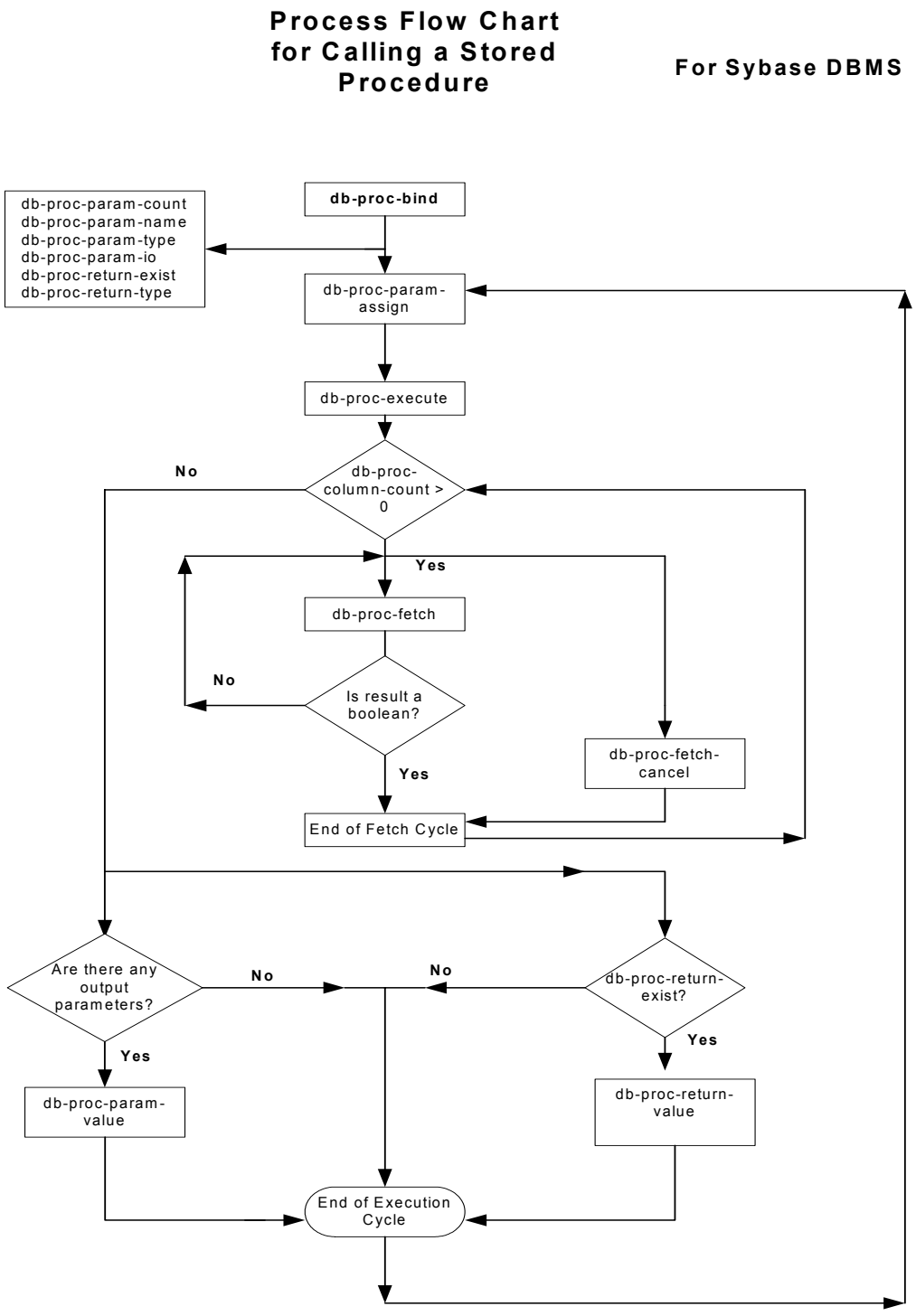


Figure 83 Dynamic Statement Flow Chart

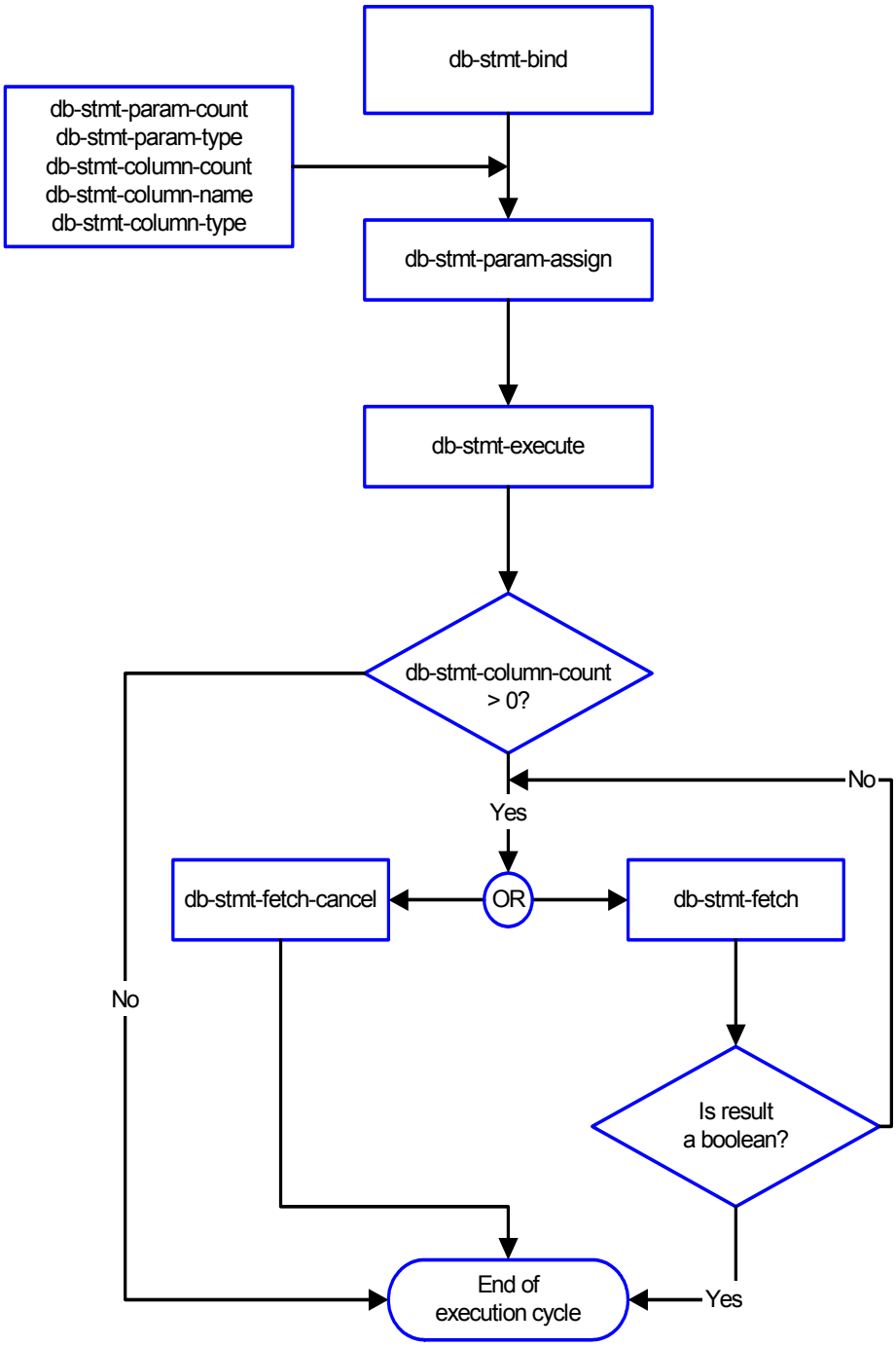
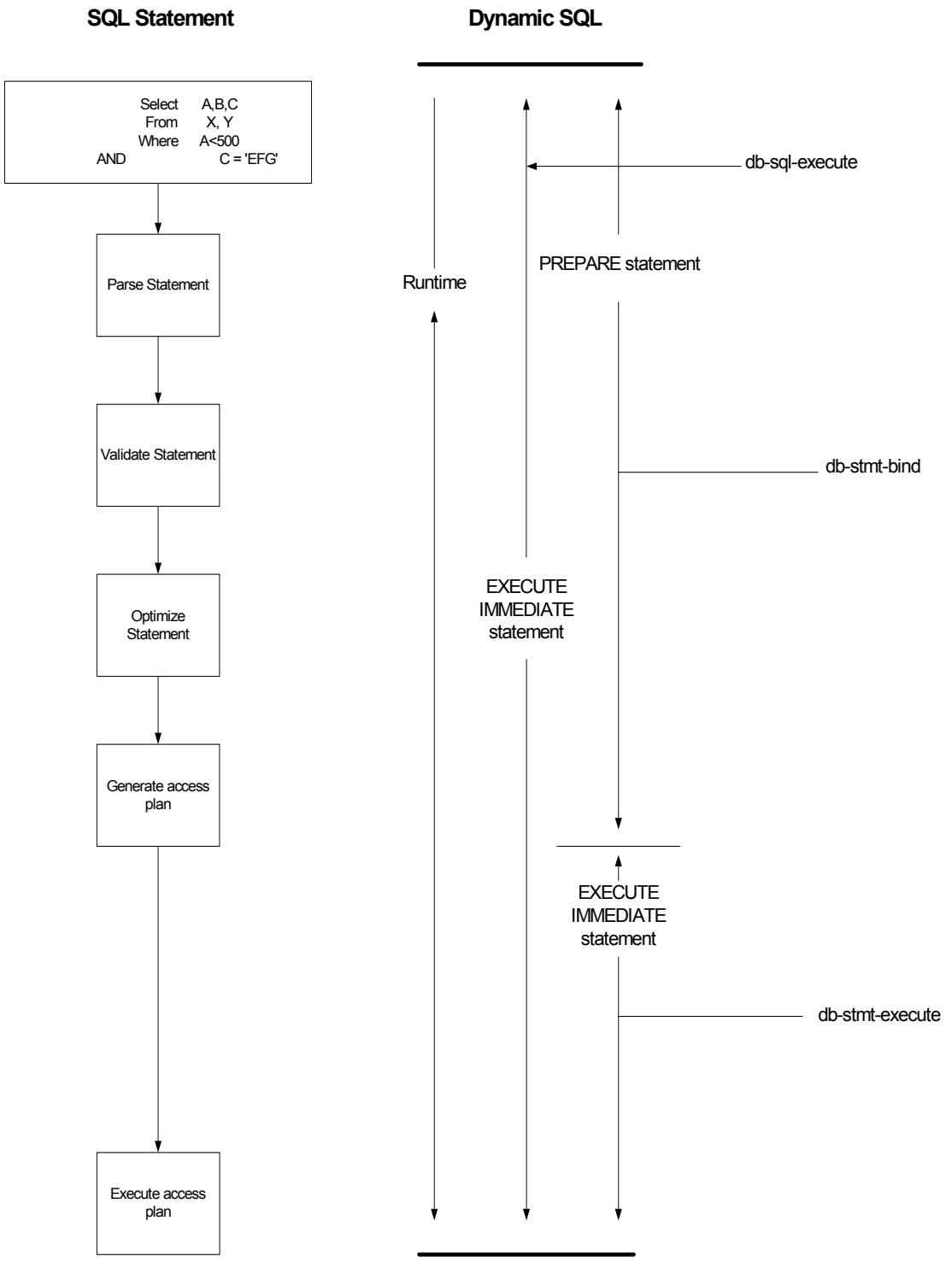


Figure 84 Example of Dynamic SQL processing



ODBC SQL Type Support

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

Table 19 ODBC SQL Type Support

SQL Type Name	SQL Datatype	Oracle Datatype
SQL_BIT	BIT	N/A
SQL_BINARY	BINARY (n)	N/A
SQL_VARBINARY	VARBINARY (n)	RAW (n)
SQL_CHAR	CHAR (n)	CHAR (n)
SQL_VARCHAR	VARCHAR (n)	VARCHAR2 (n)
SQL_DECIMAL	DECIMAL (p, s)	NUMBER (p, s)
SQL_NUMERIC	NUMERIC (p, s)	N/A
SQL_TINYINT	TINYINT	+
SQL_BIGINT	BIGINT	+
SQL_SMALLINT	SMALLINT	+
SQL_INTEGER	INTEGER	+
SQL_REAL	REAL	*
SQL_FLOAT	FLOAT(p)	FLOAT(b)
SQL_DOUBLE	DOUBLE PRECISION	FLOAT
SQL_DATE	DATE	N/A
SQL_TIME	TIME	N/A
SQL_TIMESTAMP	TIMESTAMP	DATE
SQL_LONGVARCHAR	LONG VARCHAR	LONG
SQL_LONGVARBINARY	LONG VARBINARY	LONG RAW

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

+Oracle uses number (p) to define datatypes that span TININT, BIGINT, SMALLINT, and INTEGER. Oracle int type is internally mapped to NUMBER (38) which will be returned as SQL_DECIMAL.

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

SQL_DECIMAL and SQL_NUMERIC datatypes require specification of scale which indicates the number of digits to the right of the decimal point.

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

A string

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

Boolean

If the selection-name 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-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 will display 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.

A 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**. Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define selection "select * from person where title= 'manager'")
        (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 will display 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

A 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 0)) ((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)
  )
```

```
)
```

Explanation

This example shows that after issuing a successful SQL SELECT statement, the program will loop 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

1. 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.
2. If the value of the column contains the comma (,) character, the user will be 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.

db-sql-execute

Syntax

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

Description

db-sql-execute executes 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) if successful; otherwise, returns **#f** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
...
(if (db-login hdbc "Payroll" "James" pwd)
   (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 will insert a record into the table "employee."

Notes

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

The **db-sq.-execute** function can no longer be used to commit and roll back transactions. Instead, use **db-commit** or **db-rollback**.

Note: *The Merant ODBC drivers limit the size of the SQL statement to 32 Kbyte.*

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

A 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-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 will be formatted. If the *data-string* is an empty string, the procedure will return a NULL value for all SQL datatypes 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	An SQL datatype string, i.e., SQL_VARCHAR.

Return Values

A 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" pwd)
    (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.

Notes

1. For SQL_CHAR and SQL_VARCHAR (SQL datatypes) *db-sql-format* will place 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.
2. 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"
   ```

   For SQL_CHAR and SQL_VARCHAR (SQL datatypes) **db-sql-format** will place 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.

   The following table shows the typical *data-string* and the corresponding results of the formatting for the OBDC e*Way.

Table 20 SQL Statement Format

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_DATE	1998-02-19	{d ‘1998-02019’}
SQL_DATE	19980219	{d ‘1998-02-19’}
SQL_TIME	12 :34:56	{t ‘12:34:56’}
SQL_TIME	1234	{t ‘12:34:00’}
SQL_TIMESTAMP	1998-02-19 12:34:56.789	{ts ‘1998-02-19 12:34:56.789’}

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) 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
        (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.

Note: *The Merant ODBC drivers limit the size of the SQL statement to 32 Kbyte.*

6.5 Dynamic SQL Functions

The function sin this category control the e*Way's interaction with dynamic SQL commands. For information about the differences between static and dynamic SQL functions, see **"Static vs. Dynamic SQL Functions" on page 387**.

The dynamic SQL functions are:

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 should be input or output parameters with 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** (false).Use **db-get-error-str** to retrieve the error message.

Throws

None.

Additional Information

If the user needs to input /output binary data in the raw (binary) format, they should use db-stmt-bind-binary.

Notes

1. Oracle OCI API is unable to report the datatype for each bound parameter in a dynamic statement. All bound parameters will default to VARCHAR datatypes. This will allow Oracle to implicitly convert the data string of each parameter into the correct data value of the parameter at the execution of the dynamic statement.
2. If the user needs to select the long datatype column, the long column should appear at the end of the selection list.

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 will be 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-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	string	The statement handle that identifies the dynamic statement specified. This is the handle produced by db-stmt-bind.

Return Values

A number

Returns a number greater than zero (0) when the record set is available.

Boolean

If no record set is available, the return value will be **#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	statement handle	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

A 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 returns 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 dynamic statement specified.
index	integer	An integer equal to -- 0 to db-stmt-column-count minus 1.

Return Values

A 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-execute

Syntax

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

Description

db-stmt-execute executes the dynamic statement of a specified statement-handle.

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-stmt-bind.

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	string	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 no more records are available to "fetch."

Boolean

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	string	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-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-stmt-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-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-stmt-bind.

Return Values

An Integer

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

Boolean

If unsuccessful, 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-stmt-bind.
index	integer	The number between 0 and db-stmt-param-count minus 1.

Return Values

A 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.

db-stmt-row-count

Syntax

```
(db-stmt-row-count connection-handle statement-handle index)
```

Description

db-stmt-row-count returns 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	statement handle	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

Boolean

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

Throws

None.

6.6 Stored Procedure Functions

The functions in this category control the e*Way's interaction with stored procedures.

The stored procedure functions are:

Benefits of Stored Procedures

When a 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 procedure.

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 a proc-handle if successful; otherwise, returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

Examples

```
(define hstmt (db-proc-bind hdbc "test")
(if (not (statement-handle? hstmt)
    (display "fail to bind stored procedure test\n")
)
```

Notes

ODBC does not recognize any procedure inside the PACKAGE of the Oracle DBMS. Therefore, you cannot bind any procedure defined inside the PACKAGE of the Oracle DBMS.

The procedure name is limited to 30 characters.

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 format of the input and output data is binary.

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

A string

Returns a statement-handle when successful; otherwise

Boolean

Returns **#f** (false). Use **db-get-error-str** to retrieve the error message.

Throws

None.

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

A number

Returns a number greater than zero (0) when the record set is available.

Boolean

If no record set is available, the return value will be **#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)
)
```

Notes

1 The Oracle procedure to return the result set is given here:

```
oracle_odbc.sql
```

2 The ODBC configuration parameter to set up the return result set must show

```
ProcedureRetResult = 1
```

For more information see **"Sample .odbc.ini File" on page 29**.

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

A 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)
    )
```

Notes

Since the result set of a stored procedure is returned through the parameters of the PL/SQL table type, the name of the table type parameter will be returned.

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	string	SQL datatype of the specified column in the record set --0 to db-proc-column-count minus 1.

Return Values

A 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)
    )
```

Notes

Since the result set of the stored procedure is returned through the parameters of the PL/SQL table type, a PL/SQL table can only contain one standard Oracle datatype.

db-proc-execute

Syntax

```
(db-proc-execute  connection-handle  statement-handle)
```

Description

db-proc-execute executes out 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 will fail if the exponential part of the value is larger than 38.

For example:

- 1.555E+38 is acceptable
- 1.55E+39 will prevent 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

A 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

```
(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)
    )
```

Notes

1. The Oracle procedure to return result set is given here:

   ```
   oracle_odbc.sql
   ```

2. ODBC configuration parameter to set up the capability of return result set must show

   ```
   ProcedureRetResult = 1
   ```

   For more information see Sample **.odbc.ini** **"Sample .odbc.ini File" on page 29**.

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

```
(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)
)
```

db-proc-param-assign

Syntax

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

Description

db-proc-param-assign "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 dynamic statement 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 Scenario #2 on above.

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

A 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))
)
```

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 dynamic statement 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

A 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-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.
param-index	integer	The number between 0 and db-proc-param-count minus 1.

Return Values

A 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 dynamic statement 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

A 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-value

Syntax

```
(db-proc-param-value connection-handle statement-handle param-index)
```

Description

db-proc-param-value is used to retrieve 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 dynamic statement 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

A 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)
)
```

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 dynamic statement 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))
          )
      ...
      ...
  )
```

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

A 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)
  )
```

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

A 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

1. 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. The SQL Server has a defined set of return values; or users can define their own return values.
2. The SQL Server reserves 0 to indicate a successful return, and negative values in the range of -1 to -99 are assigned to a listing of reasons for failure. Numbers 0 and -1 to -14 are in use currently (see below).

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

6.7 Message Event Functions

The functions in this category control the e*Way's message Event operations.

The message Event functions are:

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 DART Event Type Definition, retrieves all procedure 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.
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-error-str** to retrieve the error message.

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-stmt-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-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 an 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 in 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-error-str** to retrieve 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-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 SQLSELECT 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.

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 in 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-error-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 Event Definition 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-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 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 in 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-error-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)
             ($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 Collaboration Rules file 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, the Collaboration procedure 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.

Notes

1. 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**.
2. 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.

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 in 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; otherwise, returns **#f** (false). Use **db-get-error-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].

6.8 Sample Monk Scripts

This section includes sample Monk scripts which demonstrate how to use the ODBC e*Way's Monk functions. These Monk scripts demonstrate the following activities:

6.8.1 Initializing Monk Extensions

The sample script shows how to initialize the Monk extensions. This function is used by many of the other sample Monk scripts shown in this chapter.

To use this sample script in an actual implementation, modify the following values:

- **EGATE** – This designates the location of the e*Gate client.
- **dsn** – This is he name of the data source.
- **uid** – This is the user name.
- **pwd** – This is the login password.

```
;demo-init.monk

(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 STCDB variables, data source, user ID, and password
;;

(define STCDB "ORACLE8")

(load-library "stc_dbmonkext.dll")

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

6.8.2 Calling Stored Procedures

This script gives an example of calling Stored Procedures. See **"Stored Procedure Functions" on page 417** for more details.

```
;demo-proc-execute.monk

; 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 "\ndatabase login succeed !\n")

      ; bind the stored procedure
      (define hstmt1 (bind-procedure hdbc "PERSONNEL.GET_EMPLOYEES"))

      ; call the stored procedure if the binding is successful
      (if (statement-handle? hstmt1)
         (begin
            (display "call PERSONNEL.GET_EMPLOYEES to get all sales
...\n\n")
            (if (and
                  (db-proc-param-assign hdbc hstmt1 0 "30")
                  (db-proc-param-assign hdbc hstmt1 1 "10")
                )
               (execute-procedure hdbc hstmt1)
               (display (db-get-error-str hdbc))
            )
         )
      )
```

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

6.8.3 Inserting Records with Dynamic SQL Statements

```
;demo-stmt-insert.monk

; 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 SCOTT.BONUS SELECT ENAME, JOB, SAL, COMM
FROM SCOTT.EMP WHERE DEPTNO = ?")

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

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
           (display "\nInsert accounting department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "10")
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the insertions ...\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 "\nInsert sales department into bonus table
...\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "20")
               (if (execute-statement hdbc hstmt1)
                  (begin
                     (display "\nCommit the insertions ...\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))
         )
```

6.8.4 Updating Records with Dynamic SQL Statements

```
;demo-stmt-update.monk

; 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 SCOTT.BONUS SET COMM = ? WHERE JOB = ?")

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

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nUpdate commission of manager ...\n")
            (if
               (and
                  (db-stmt-param-assign hdbc hstmt1 0 "10")
                  (db-stmt-param-assign hdbc hstmt1 1 "MANAGER")
               )
               (if (execute-statement hdbc hstmt1)
                  (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 "\nUpdate commission of clerk ...\n")
            (if
               (and
                  (db-stmt-param-assign hdbc hstmt1 0 "20")
                  (db-stmt-param-assign hdbc hstmt1 1 "CLERK")
               )
               (if (execute-statement hdbc hstmt1)
                  (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))
                     )
                  )
               )

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

6.8.5 Selecting Records with Dynamic SQL Statements

```
;demo-stmt-select.monk

; 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 EMPNO, ENAME, JOB FROM SCOTT.EMP WHERE JOB = ?")
(define stmt2 "SELECT ENAME, DNAME, JOB, HIREDATE FROM SCOTT.EMP,
SCOTT.DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO AND DEPT.DNAME = ?")

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

      ; 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 all salesman ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "SALESMAN")
               (if (not (execute-statement hdbc hstmt1))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
            (display "\nList all manager ...\n\n")
            (if (db-stmt-param-assign hdbc hstmt1 0 "MANAGER")
               (if (not (execute-statement hdbc hstmt1))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
         )
      )

      (if (statement-handle? hstmt2)
         (begin
            (display "\nList employee of accounting department
...\n\n")
            (if (db-stmt-param-assign hdbc hstmt2 0 "ACCOUNTING")
               (if (not (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))
         )
```

6.8.6 Deleting Records with Dynamic SQL Statements

```
;demo-stmt-delete.monk

; 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 SCOTT.BONUS WHERE ENAME IS NOT NULL")

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

      ; bind the dynamic statement
      (define hstmt1 (bind-statement hdbc stmt1))

      ; assign parameter and execute the dynamic statement
      (if (statement-handle? hstmt1)
         (begin
            (display "\nDelete records from scott.bonus table ...\n")
            (if (execute-statement hdbc hstmt1)
               (begin
                  (display "\nCommit the deletions ...\n")
                  (if (not (db-commit 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))
)
```

6.8.7 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 **"Static SQL Functions" on page 387** and **"Dynamic SQL Functions" on page 404** for more details.

```
;demo-image-insert.monk

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

(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 "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 (display-stmt-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 (bind-image-statement hdbc stmt)
   (let ((hstmt (db-stmt-bind-binary hdbc stmt)))
      (display (string-append "\nDynamic statement : " stmt "\n"))
      (if (statement-handle? hstmt)
         (begin
;            (db-stmt-param-bind hdbc hstmt 0 "SQL_INTEGER" 4 0)
;            (db-stmt-param-bind hdbc hstmt 1 "SQL_VARCHAR" 20 0)
;            (db-stmt-param-bind hdbc hstmt 2 "SQL_VARCHAR" 10 0)
;            (db-stmt-param-bind hdbc hstmt 3 "SQL_INTEGER" 38 0)
;            (db-stmt-param-bind hdbc hstmt 4 "SQL_INTEGER" 38 0)
;            (db-stmt-param-bind hdbc hstmt 5 "SQL_INTEGER" 10 0)
            (db-stmt-param-bind hdbc hstmt 6 "SQL_LONGVARBINARY"
2000000 0)
            (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
   )
)


(define image1-id "7100")
(define image1-name "Coast")
(define image1-type "JPEG")
(define image1-width "1280")
(define image1-height "1024")
(define image1-file (string-append image1-name ".jpg"))

(define image-port (open-input-file image1-file))
(define image1-data (read image-port 1000000))
(close-port image-port)
(define image1-size (number->string (string-length image1-data)))

(define image2-id "7200")
(define image2-name "Launch")
(define image2-type "JPEG")
(define image2-width "2000")
(define image2-height "1600")
(define image2-file (string-append image2-name ".jpg"))

(define image-port (open-input-file image2-file))
(define image2-data (read image-port 2000000))
(close-port image-port)
(define image2-size (number->string (string-length image2-data)))

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

(define stmt0 "select count(0) from SCOTT.IMAGE where PIX_ID = ?")
(define stmt1 "insert into SCOTT.IMAGE (PIX_ID, PIX_NAME, PIX_TYPE,
BYTE_SIZE, PIX_WIDTH, PIX_HEIGHT, PIX_DATA) values (?, ?, ?, ?, ?, ?,
?)")

(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-max-long-data-size hdbc 2000000)) (newline)

; bind the query and insert statement
(define hquery  (bind-statement hdbc stmt0))
(define hinsert (bind-image-statement hdbc stmt1))

(if (and
   (statement-handle? hquery)
   (statement-handle? hinsert)
 )
 (begin
   (if (not (query-exist hdbc hquery image1-id))
     (begin
       (display (string-append "insert image " image1-file "\n"))
       (if (and
          (db-stmt-param-assign hdbc hinsert 0 image1-id)
          (db-stmt-param-assign hdbc hinsert 1 image1-name)
          (db-stmt-param-assign hdbc hinsert 2 image1-type)
          (db-stmt-param-assign hdbc hinsert 3 image1-size)
          (db-stmt-param-assign hdbc hinsert 4 image1-width)
          (db-stmt-param-assign hdbc hinsert 5 image1-height)
          (db-stmt-param-assign hdbc hinsert 6 image1-data)
         )
         (if (execute-statement hdbc hinsert)
          (db-commit hdbc)
          (display (db-get-error-str hdbc))
         )
         (display (db-get-error-str hdbc))
       )
     )
   )

   (if (not (query-exist hdbc hquery image2-id))
     (begin
       (display (string-append "insert image " image2-file "\n"))
       (if (and
          (db-stmt-param-assign hdbc hinsert 0 image2-id)
          (db-stmt-param-assign hdbc hinsert 1 image2-name)
          (db-stmt-param-assign hdbc hinsert 2 image2-type)
          (db-stmt-param-assign hdbc hinsert 3 image2-size)
          (db-stmt-param-assign hdbc hinsert 4 image2-width)
          (db-stmt-param-assign hdbc hinsert 5 image2-height)
          (db-stmt-param-assign hdbc hinsert 6 image2-data)
         )
         (if (execute-statement hdbc hinsert)
          (db-commit 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))
)
```

6.8.8 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 **"Static SQL Functions" on page 387** and **"Dynamic SQL Functions" on page 404** for more details.

```
;demo-image-select.monk

; 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 "BITMAP") (set! file_name (string-append
first_name ".bmp")))
         ((string=? file_type "TIFF") (set! file_name (string-append
first_name ".tif")))
         (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 stmt "select PIX_NAME, PIX_TYPE, PIX_WIDTH, PIX_HEIGHT,
PIX_DATA from SCOTT.IMAGE where PIX_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-max-long-data-size hdbc 2000000)) (newline)

      ; bind the select statement
      (define hselect (bind-binary-statement hdbc stmt))

      ; execute the dynamic statement
      (display "select IMAGE table\n")
      (if (statement-handle? hselect)
         (begin
            (if (db-stmt-param-assign hdbc hselect 0 "7100")
               (if (not (execute-statement hdbc hselect))
                  (display (db-get-error-str hdbc))
               )
               (display (db-get-error-str hdbc))
            )
            (if (db-stmt-param-assign hdbc hselect 0 "7200")
               (if (not (execute-statement hdbc hselect))
                  (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))
)
```

6.8.9 Common Supporting Routines

This sample script displays and defines values and parameters for stored procedures. The routines contained in this script are used by many of the Monk samples in this chapter. For more details about functions used in this script, see **"Stored Procedure Functions" on page 417**

```
;demo-common.monk

;;
;; 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
   )
)
```

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Z