

Daffodil DB ODBC Driver Reference Guide

Version 1.0

July 2004

Copyright © Daffodil Software Limited
Sco 42, 3rd Floor
Old Judicial Complex, Civil Lines
Gurgaon - 122001
Haryana, India.
www.daffodildb.com

All rights reserved. Daffodil DB™ is a registered trademark of Daffodil Software Limited. Java™ is a registered trademark of Sun Microsystems, Inc. All other brand and product names are trademarks of their respective companies.

About This Guide

This guide can be used by developers for getting acquainted with the Installation, syntax and usage details of Daffodil DB ODBC API (Application Programmable Interfaces) to access Daffodil DB data stores on Windows 98 and advanced windows clients.

Target Audience

This reference guide is designed for developers who want to develop ODBC-compliant client applications on Windows 98 and advanced windows clients to access data from Daffodil DB data sources. It can also be used by end users who use existing ODBC-compliant Windows applications to connect to Daffodil DB data sources.

Guidelines For Using This Guide

The information in this guide is organized as follows

Chapter1 **“Introduction,”** contains platform-specific information about system requirements. It also describes the ODBC data types supported by Daffodil DB ODBC Driver

Chapters 2, **“Installing Procedure,”** provide details about the installation procedure for Windows 98 and advanced windows clients.

Chapter 3, **“Configuration Details,”** explain how to configure the ODBC Driver and how to connect to a Daffodil DB data source.

Chapter 4, **“ODBC API Details,”** lists the ODBC API functions supported by the DaffodilDB ODBC Driver”

Appendix A, **“Diagnostics,”** contains a list of errors.

Other Sources of Information

To get more information on Daffodil DB ODBC Driver, users can get in touch with us at support@daffodildb.com

Table of Contents

CHAPTER 1 – INTRODUCTION	6
System Requirements.....	6
Supported Data Types.....	7
Download instructions	8
Chapter 2 – Installation Procedure	9
Chapter 3 – Configuration Details	10
Daffodil DB Embedded Edition.....	12
Daffodil DB Server Edition	13
Chapter 4 – ODBC API Functions	15
SQLAllocHandle.....	15
SQLSetEnvAttr	17
SQLSetConnectAttr	18
SQLGetInfo.....	19
SQLGetFunctions	20
SQLConnect.....	21
SQLDriverConnect	23
SQLSetStmtAttr	25
SQLGetEnvAttr	26
SQLGetConnectAttr	27
SQLGetStmtAttr	28
SQLPrepare.....	29
SQLNumParams	30
SQLBindParameter	31
SQLExecute	32
SQLExecDirect.....	33
SQLNativeSql.....	34
SQLDescribeParam.....	35
SQLRowCount.....	36
SQLNumResultCols	37
SQLDescribeCol	38
SQLColAttribute.....	40
SQLBindCol	41
SQLFetch	42
SQLExtendedFetch.....	43
SQLFetchScroll.....	44
SQLGetData.....	45
SQLEndTran.....	46
SQLTables	47
SQLTablePrivileges.....	49
SQLColumns.....	51
SQLColumnPrivileges	53

SQLPrimaryKeys	55
SQLForeignKeys	57
SQLGetTypeInfo	59
SQLProcedures	61
SQLProcedureColumns	63
SQLGetDiagRec	65
SQLCancel.....	67
Appendix A – Diagnostics: ODBC SQL States and their Description.....	68

CHAPTER 1-INTRODUCTION

The Daffodil DB ODBC Driver complies with the Microsoft Open Database Connectivity (ODBC) specifications. ODBC is an application programming interface (API) that allows applications to access database management systems using Structured Query Language (SQL). By adding the Daffodil DB ODBC driver, User can link an application to Daffodil DB. Connecting to a data source means establishing connection with Daffodil DB to access the data. When you connect to a data source from an application through Daffodil DB ODBC driver, the driver makes the connection for you, either locally or across a network.

Note: Daffodil DB ODBC driver can be downloaded in .zip format from www.daffodildb.com.

System Requirements

Table 1 summarizes the system requirements for Daffodil DB ODBC Driver.

S.No	Type Of Resource	Description
1	Daffodil DB version	Daffodil DB v 2.5 and above
2	Operating System	Windows 98 and above
3	Required Secondary Memory	750 KB on Hard Disk Drive
4	RAM Usage	16 MB
5	Type Of Driver	32- Bit-ODBC
6.	Number of Connections and Statements Supported	Multiple Connections and Multiple Statements per connection.

Table 1

Supported Data Types

Table 2 shows how Daffodil DB data types are mapped to standard ODBC data types.

S.No	Daffodil DB Data Type	ODBC Data Type
1	Date	SQL_TYPE_DATE
2	Time	SQL_TYPE_TIME
3	Timestamp	SQL_TYPE_TIMESTAMP
4	Boolean	SQL_C_BIT
5	Tinyint ,byte	SQL_C_TINYINT
6	Binary , varbinary , or other binary type	SQL_C_BINARY
7	char , varchar, varchar2	SQL_C_CHAR
8	Float	SQL_C_FLOAT
9	Short, smallint	SQL_C_HORT
10	int , integer	SQL_C_LONG
11	Long	SQL_C_BIGINT
12	Double, double precision	SQL_C_DOUBLE

Table 2

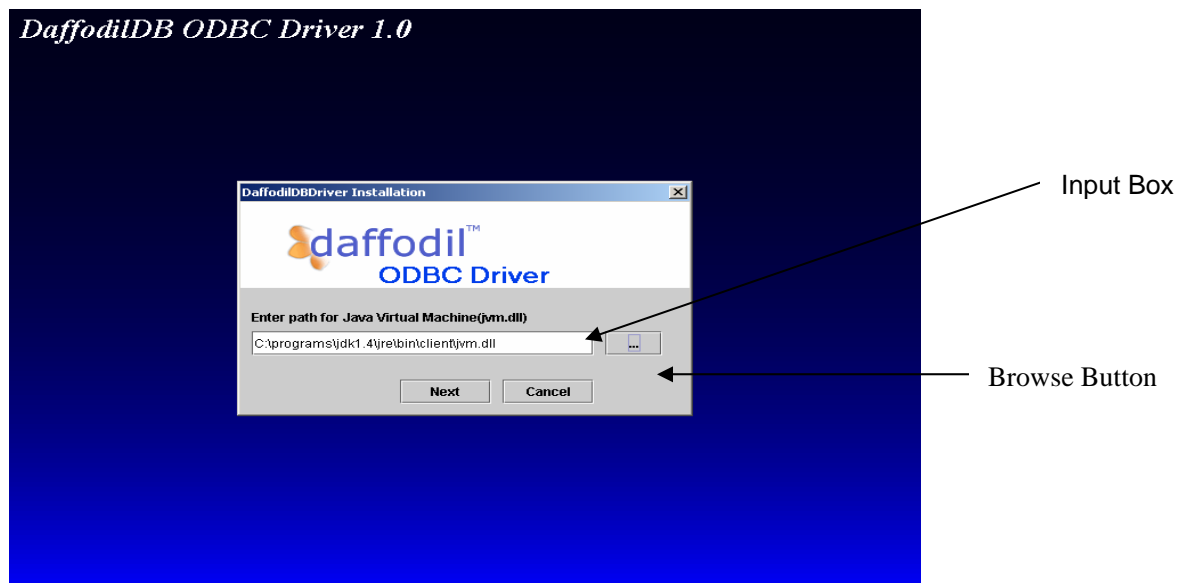
Download instructions


Daffodil DB ODBC Driver is available for download at www.daffodildb.com. The downloaded zip file contains two files namely **DaffodilODBCDriver.jar** and **ReadMe.txt**. Among these files, **DaffodilODBCDriver.jar** is used to install Daffodil DB ODBC Driver whereas **ReadMe.txt** provides the installation details. For a detailed description on Installation instruction please refer to chapter 2 of this guide.

Chapter 2 – Installation Procedure

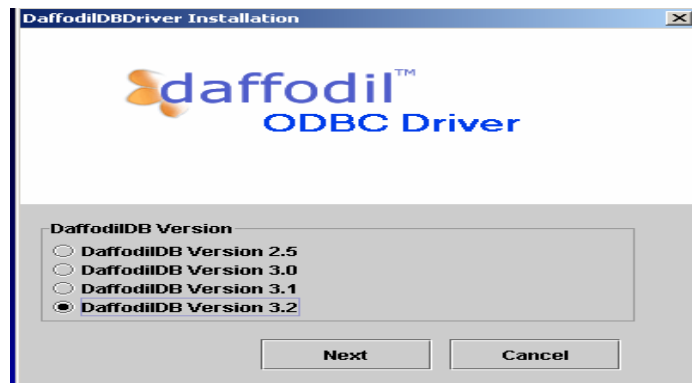
Daffodil DB ODBC driver can be installed with the help of a simple and easy to follow process. This chapter provides comprehensive details about the Installation procedure. The stepwise procedure is explained as follows.

Step1. On executing DaffodilODBCDriver.jar; following screen will appear.



The users shall provide the path for Java Virtual Machine dynamic linked library (jvm.dll file) inside the space provided and then click the next button. The path can be browsed with the help of  browse button.

Step 2: The next screen will prompt users to provide information about Installed version of Daffodil DB. Choose appropriate version and click next.



Step 3: Restart the Computer to finalize the installation.

Chapter 3 – Configuration Details

A DSN is used for accessing the Database. It is necessary to create a DSN for connecting to the Daffodil DB Database with the help of Daffodil DB ODBC Driver. The procedure to create a DSN is as follows,

Step 1. Follow the path provided below as per the Installed windows version.

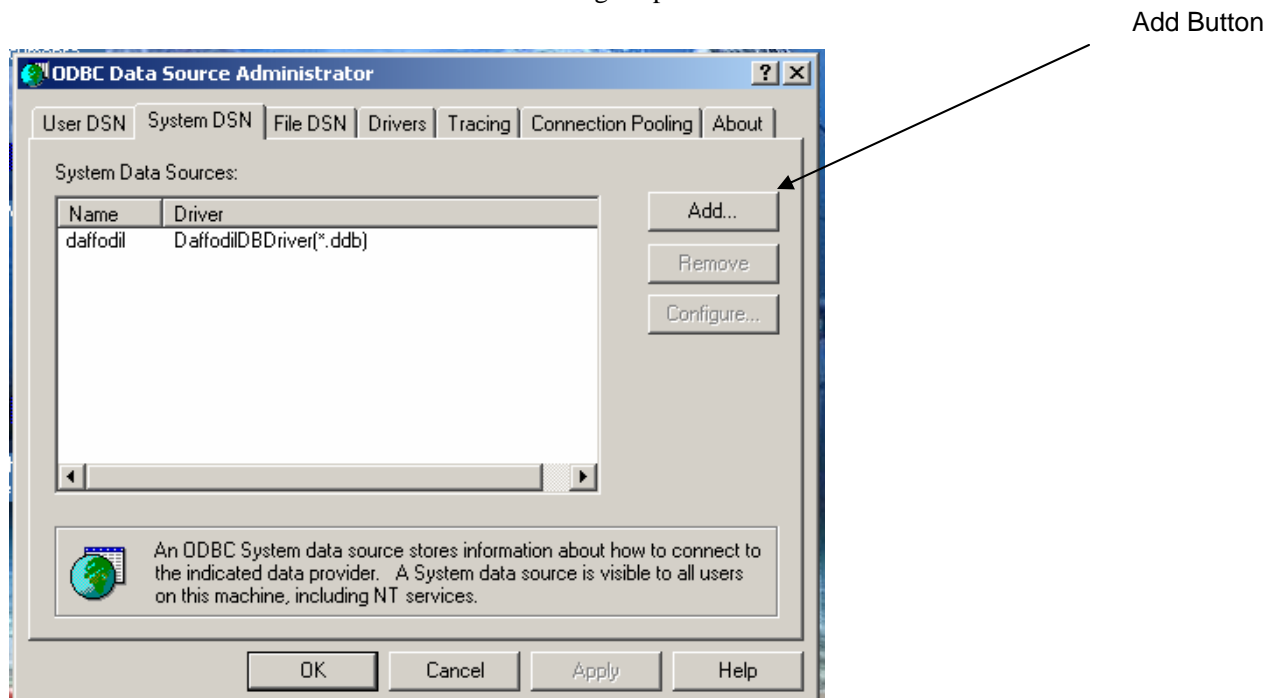
For Windows 98

Go to, Settings(Control Panel(32 – Bit ODBC and choose type of DSN.

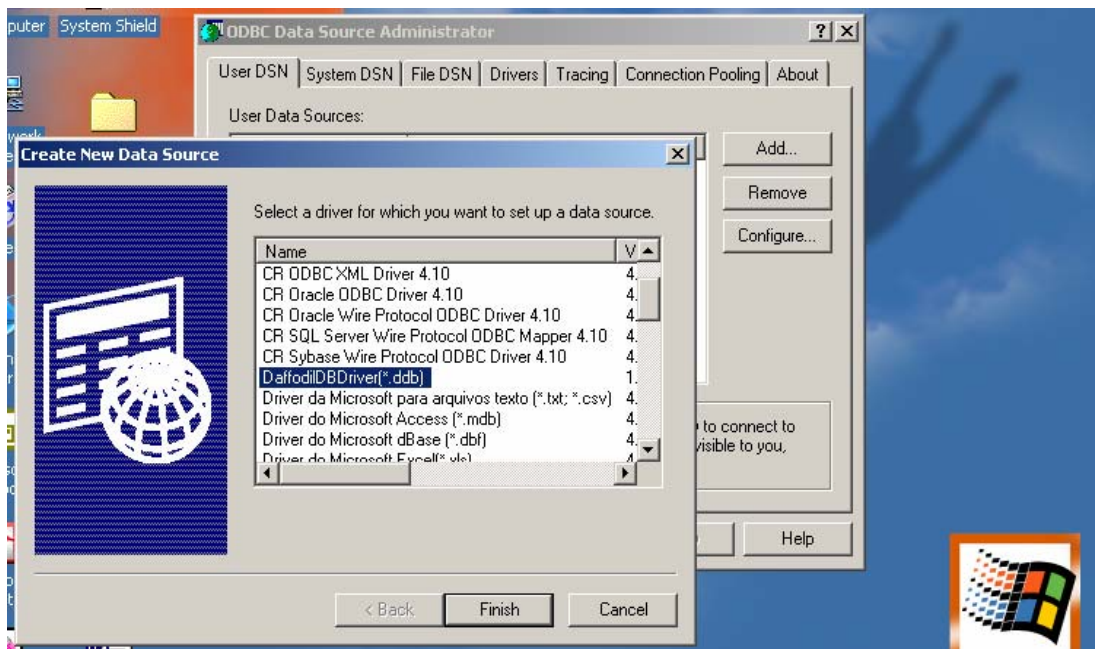
For advance windows versions (Windows ME, XP, Windows2000)

Go to, Settings(Control Panel(Administrative Tools(Data Sources (ODBC) and choose type of DSN.(System or User DSN).

Step 2. ODBC Data Source Administrator Screen will be displayed. To create a new DSN, Click on the Add Button as shown in the following snapshot.



Step 3. Create New Data Source screen prompts for the type of data source. Choose DaffodilDBDriver(*.ddb) as shown in the following Snapshot and click on the finish Button.



Daffodil DB ODBC driver can be used for connecting to Daffodil DB Embedded as well as Daffodil DB Server data sources. The process of configuring a User DSN as per type of Installation has been explained as follows,

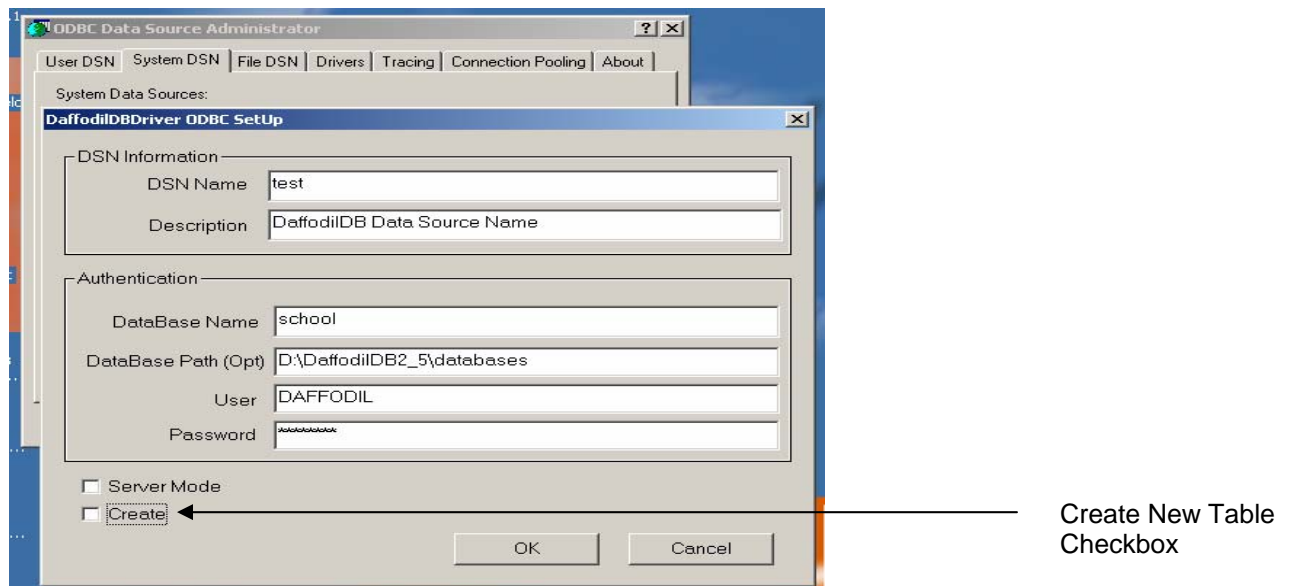
Daffodil DB Embedded Edition

Once the above listed steps for creating a DSN have been performed, Daffodil DB ODBC Setup screen prompts for some basic information like DSN name, Description, Database name, Path and the user information, which is used for connecting to Daffodil DB data sources. The description about the user information has been included in table 4.

S. No	Attribute	Description
1	Data Source Name	A string that identifies this Daffodil DB data source configuration.
2	Description	An optional long description of a data source name.
3	Database Name	If the DSN is made for connecting to an existing database.
4	Database Path (Optional)	Optional parameter, If this parameter is not specified, by default the path of Daffodil DB will be used.
5	User Information	Valid Username and Password.

Table 4

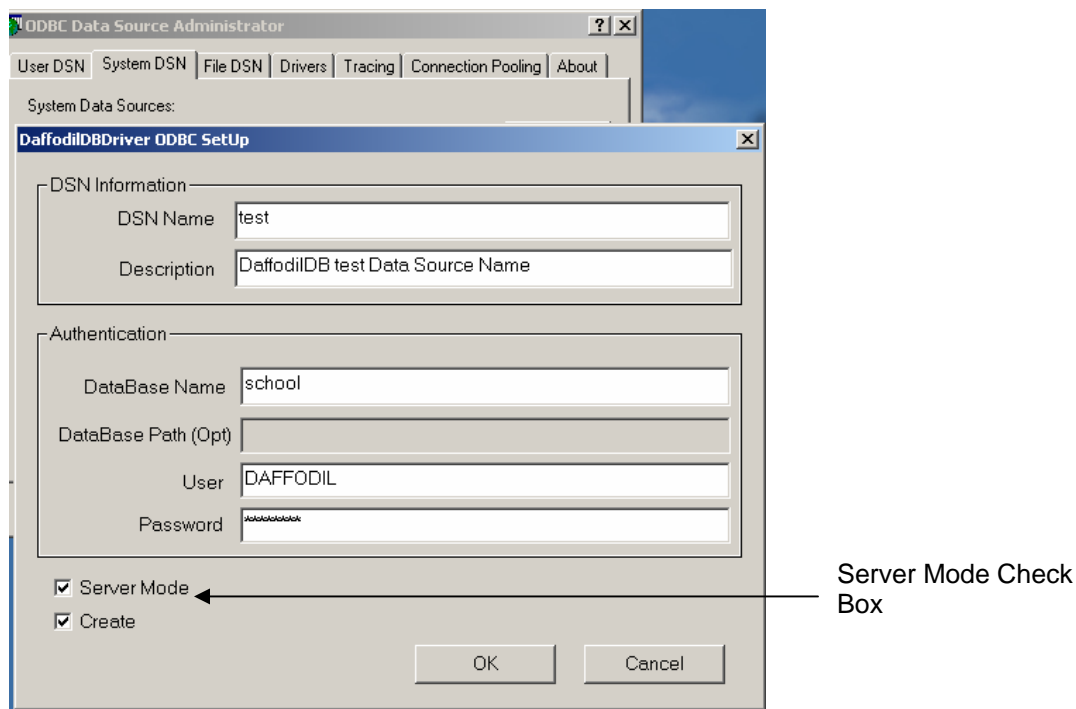
Note: To allow the creation of a new Database, users can select the Create checkbox. It is depicted in the following snapshot.



Daffodil DB Server Edition

Users shall follow these simple and easy to follow steps for creating and testing a Daffodil DB Server DSN. These steps shall be followed in a sequential manner

Step A: Follow step 1, 2 and 3. Provide various details listed in Table 4 in the space provided for it in “Daffodil DB ODBC Driver” screen as displayed in the following snapshot.

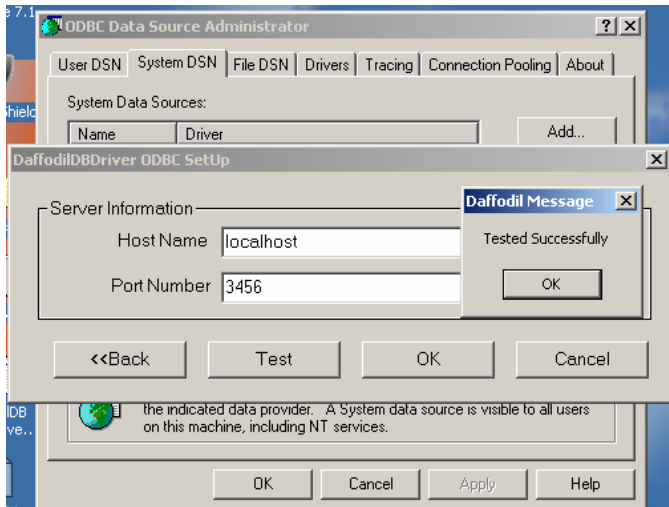


Step B: Check the Check Box labeled as Server Mode as depicted in the figure above.

Step C: Click OK.

Step D: Users will be guided to the Daffodil DB Driver ODBC Setup screen. Provide basic information like Host Name and Port Number to test the connection to the specified data source as shown in the following snapshot.

Note: For testing the connection, localhost shall be specified as the Host Name with 3456 as the Port Number (By Default Daffodil DB runs on this port).



If users can see the message “Tested Successfully” as depicted in the snapshot above, then their DSN has been created and tested successfully. Now this DSN can be used for connecting to Daffodil DB Data sources. One ODBC DSN can be used to connect to multiple Daffodil DB data sources.

Chapter-4: ODBC API Functions

ODBC API Functions are useful for performing fast operations on database. This chapter is designed to help the developers in implementing the functionality provided by ODBC API's.

In any ODBC application, users first need to set up the ODBC environment and connect to a data source before executing any SQL statement. Similarly, users shall disconnect from the database and free the allocated memory for the ODBC environment on program termination.

SQLAllocHandle

Summary

SQLAllocHandle allocates an environment, connection, statement; you must use the **SQLAllocHandle** function to set up the ODBC environment before you can call any other ODBC functions. When you call the **SQLAllocHandle** function, the Daffodil DB driver allocates an area of memory for environment information and returns a handle to your application. This handle is known as the environment handle.

Syntax

```
SQLRETURN SQL_API SQLAllocHandle (SQLSMALLINT HandleType,  
                                  SQLHANDLE InputHandle,  
                                  SQLHANDLE *OutputHandlePtr);
```

Arguments

HandleType [Input]: Following handle types are supported,

- 1.SQL_HANDLE_ENV
- 2.SQL_HANDLE_DBC
- 3.SQL_HANDLE_STMT

InputHandle [Input]: If *HandleType* is SQL_HANDLE_ENV, InputHandle shall be SQL_NULL_HANDLE and if it is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle.

OutputHandlePtr [Output]: Pointer to a buffer in which the newly allocated data structure is returned

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, or SQL_ERROR.

Example

```
SQLHENV  henv;
SQLHDBC  hdbc;
SQLRETURN  returnCode;

/*Allocate environment handle */
returnCode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
/*Allocate Connection handle */

returnCode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
/*Allocate statement handle always after connection */

returnCode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

SQLSetEnvAttr

Summary

This function is used to set attributes for managing aspects of environment handle.

Syntax

```
SQLRETURN SQL_API SQLSetEnvAttr(SQLHENV henv,  
                                SQLINTEGER Attribute,  
                                SQLPOINTER ValuePtr,  
                                SQLINTEGER StringLength);
```

Arguments

henv: Environment handle.

Attribute [Input]: This is used to set attributes like SQL_ATTR_ODBC_VERSION. DaffodilDB ODBC driver follows ODBC 3.0 compliance (SQL_OV_ODBC3).

ValuePtr [Input]: Pointer to the value to be associated with *Attribute*. Depending on the value of *Attribute*, a 32-bit integer value or pointer to a null-terminated character string will be assigned to *ValuePtr*.

StringLength [Input]: If *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *ValuePtr* is an integer, *StringLength* is ignored.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example:

```
SQLRETURN returnCode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,  
(void*)SQL_OV_ODBC3, 0);
```

SQLSetConnectAttr

Summary

This function is used to set attributes for managing aspects of connections.

Syntax

```
SQLRETURN SQL_API SQLSetConnectAttr ( SQLHDBC hdbc,  
                                       SQLINTEGER Attribute,  
                                       SQLPOINTER ValuePtr,  
                                       SQLINTEGER StringLength);
```

Arguments

hdbc : Pointer to the connection handle allocated by SQLAllocHandle.

Attribute [Input]: To set connection Attribute. i.e. SQL_ATTR_ODBC_CURSORS.

ValuePtr [Input]: It is a Pointer to the value associated with *Attribute*. Depending on the value of *Attribute*, *ValuePtr* will contain a 32-bit integer value or it will point to a null-terminated character.

StringLength [[Input]: Parameter value depends on following conditions

- If *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *ValuePtr* is an integer, *StringLength* is ignored.
- If *ValuePtr* is a pointer to a character string, then *StringLength* is the length of the string or SQL_NTS.
- If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *StringLength* should have the value SQL_IS_POINTER.
- If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_IS_UIINTEGER, as appropriate.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLRETURN returnCode = SQLSetConnectAttr(hdbc,SQL_ATTR_AUTOCOMMIT,(void*)  
SQL_AUTOCOMMIT_ON , SQL_IS_POINTER);
```

SQLGetInfo

Summary

SQLGetInfo returns general information about the driver and data source associated with a connection.

Syntax

```
SQLRETURN SQL_API SQLGetInfo( SQLHDBC hdbc, SQLUSMALLINT fInfoType,
                               SQLPOINTER rgbInfoValue,
                               SQLSMALLINT cbInfoValueMax,
                               SQLSMALLINT *pcbInfoValue);
```

Arguments

hdbc : Connection handle.

fInfoType [Input]: Type of information. I. e. SQL_MAX_SCHEMA_NAME_LEN.

rgbInfoValue [output]: Pointer to a buffer in which to return the information. Depending on the *fInfoType* requested, the information returned may contain:
a null-terminated character string,
A SQLUSMALLINT value,
a SQLINTEGER bit mask, an SQLINTEGER flag,
a SQLINTEGER
binary value.

cbInfoValueMax [Input]: Length of the * or if **rgbInfoValue** buffer. If the value in * or if **rgbInfoValue** is not a character string *BufferLength* argument is ignored.

pcbInfoValue [Output]: Pointer to a buffer in which to return the total number of bytes available to return in * **rgbInfoValue** . If value returned in **rgbInfoValue** is not a string value, it may be SQLSMALLINT OR SQLINTEGER.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLRETURN retCode=
SQLGetInfo(hdbc,SQL_MAX_CATALOG_NAME_LEN,&value,0,&buffer);
```

SQLGetFunctions

Summary

supports a specific ODBC function. This function is implemented in the Driver Manager; it can also be implemented in drivers. If a driver implements **SQLGetFunctions**, the Driver Manager calls the function in the driver. Otherwise, it executes the function itself. We function is fully supported in our driver.

Syntax

```
SQLRETURN SQL_API SQLGetFunctions ( SQLHDBC hdbc,  
                                   SQLUSMALLINT fFunction,  
                                   SQLUSMALLINT *pfExists);
```

Arguments

hdbc: Connection handle.

fFunction [Input]: A constant #define value that identifies the ODBC function of interest such as SQL_API_ODBC3_ALL_FUNCTIONS. SQL_API_ODBC3_ALL_FUNCTIONS is used by an ODBC 3.x application to determine support of ODBC 3.x and earlier functions.

pfExists [Output]: If *fFunction* identifies a single ODBC function, this output buffer contains a single value (SQL_TRUE), for driver supported functions else returns SQL_FALSE.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLRETURN retCode=SQLGetFunctions(hdbc,SQL_API_SQLALLOCHANDLE,&ret_val);
```

SQLConnect

Summary

This function establishes connections with a driver using DSN. The connection holds references storage of all information about the connection to the data source, including status, transaction state, and error information.

Syntax

```
SQLRETURN SQL_API SQLConnect (  SQLHDBC hdbc,
                                SQLCHAR *dsnName,
                                SQLSMALLINT cbDSN,
                                SQLCHAR *userName,
                                SQLSMALLINT cbUID,
                                SQLCHAR *authentication,
                                SQLSMALLINT cbAuthStr);
```

Arguments

hdbc: Pointer to the connection handle allocated by SQLAllocHandle.

dsnName [Input]: Data source name (DSN)

cbDSN [Input]: Length of *dsnName* ().

username [Input]: Name of the authenticated User.

cbUID [Input]: Length of *userName*.

Authentication [Input]: Authentication string (typically the password).

cbAuthStr [Input]: Length of *Authentication string*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLHENV  henv;
SQLHDBC  hdbc;
SQLHSTMT hstmt;
SQLRETURN  returnCode;
```

```
/*Allocate environment handle */
```

```
returnCode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```

```
if(returnCode == SQL_SUCCESS || returnCode == SQL_SUCCESS_WITH_INFO)
```

```
    /* Set the ODBC version environment attribute */
```

```
    returnCode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3,
0);
```

```
    if (returnCode == SQL_SUCCESS || returnCode == SQL_SUCCESS_WITH_INFO)
```

```
/* Allocate connection handle */
returnCode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

    if (returnCode == SQL_SUCCESS || returnCode == SQL_SUCCESS_WITH_INFO)
/* Set commit mode. */
    SQLSetConnectAttr(hdbc,                                SQL_ATTR_AUTOCOMMIT,
        (void*)SQL_AUTOCOMMIT_ON, 0);

/* Connect to data source, Assume DSN name is daffodil */
returnCode = SQLConnect(hdbc, (SQLCHAR*) "daffodil", SQL_NTS,
(SQLCHAR*) "DAFFODIL", SQL_NTS, (SQLCHAR*) "daffodil", SQL_NTS);

    if (returnCode == SQL_SUCCESS || returnCode == SQL_SUCCESS_WITH_INFO)
/* Allocate statement handle */
returnCode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

        if (returnCode == SQL_SUCCESS || returnCode == SQL_SUCCESS_WITH_INFO)

/* Process Data */
            ;
            ;
            ;

            SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        }
        SQLDisconnect(hdbc);
    }
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
}
}
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

SQLDriverConnect

Summary

This ODBC API function is used as an alternative to **SQLConnect**. It supports data sources that require more connection information than **SQLConnect** (DSN,user name, password). **SQLDriverConnect** can display dialog boxes to prompt the user for providing connection information that are not defined in the system information.

Daffodil DB ODBC Driver incorporates this function to incorporate the functionality to

- Establish a connection using a connection string that contains the data source name, user ID, password.
- Establish a connection using a partial connection string or no additional information; in this case, the Driver Manager and the driver can each prompt the user for connection information.

Syntax

```
SQLRETURN SQL_API SQLDriverConnect ( SQLHDBC hdbc,SQLHWND hwnd,  
SQLCHAR *szConnStrIn,  
SQLSMALLINT cbConnStrIn,  
SQLCHAR *szConnStrOut,  
SQLSMALLINT cbConnStrOutMax,  
SQLSMALLINT *pcbConnStrOut,  
SQLUSMALLINT fDriverCompletion);
```

Arguments

Hdbc: Connection handle.

hwnd [Input]: Window handle.

szConnStrIn [Input]: Input connection string, each information field must be separated using semicolon (;).

i.e. “DSN=daffodil;UID=daffodil;PWD=daffodil”

or

“DSN=daffodil; User=daffodil; Password=daffodil”

cbConnStrIn [Input]: Length of * cbConnStrIn in bytes.

szConnStrOut [Output]: Pointer to a buffer for the completed connection string. Upon Successful connection to the target data source, this buffer contains the completed connection string.

cbConnStrOutMax [Input]: Length of the * szConnStrOut buffer.

pcbConnStrOut [Output]: Pointer to a buffer in which to return the total number of characters available to return in ***cbConnStrOutMax** . If actual length of connection string is more then your buffer (cbConnStrOutMax) than string date will be truncated.

fDriverCompletion [Input]:Flag that indicates whether the Driver Manager or driver must prompt for more connection information. The allowed values are

- (i) SQL_DRIVER_PROMPT:

Driver prompts to show whether connection string complete or not.

- (ii) SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED.

Daffodil DB Driver checks the validity of connection string, if driver gets the necessary information it connects to the data source without prompting the user

- (iii) SQL_DRIVER_NOPROMPT.

Driver does not show any dialog boxes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLCHAR * outstr = new SQLCHAR[1024];
SQLSMALLINT outStrLen;
SQLRETURN retCode = SQLDriverConnect( hdbc.GetDesktopWindow(),
(SQLTCHAR*)"DSN=daffodil;PWD=daffodil;UID=daffodil",SQL_NTS,outstr,SQL_NTS,
&outStrLen,SQL_DRIVER_PROMPT);
```

SQLSetStmtAttr

Summary

This function is used to set statement attributes.

Syntax

```
SQLRETURN SQL_API SQLSetStmtAttr( SQLHSTMT hstmt,  
                                  SQLINTEGER Attribute,  
                                  SQLPOINTER ValuePtr,  
                                  SQLINTEGER StringLength);
```

Arguments

hstmt: Pointer to the connection handle allocated by SQLAllocHandle.

Attribute[Input]: Attribute to set such as SQL_ATTR_CONCURRENCY.

ValuePtr [Input]: Pointer to the value to be associated with *Attribute*. Depending on the value of *Attribute*, *ValuePtr* will be a 32-bit integer value or point to a null-terminated character string.

StringLength[Input]: Parameter value depends on following conditions

- If *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **Value*
- If *ValuePtr* is an integer, *StringLength* is ignored.
- If *ValuePtr* is a pointer to a character string, then *StringLength* is the length of the string or SQL_NTS.
- If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *StringLength* should have the value SQL_IS_POINTER.
- If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_IS_UIINTEGER, as appropriate.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example:

```
SQLRETURN retCode = SQLSetStmtAttr (hstmt,SQL_ATTR_CURSOR_TYPE,(void*)  
SQL_CURSOR_FORWARD_ONLY,SQL_IS_INTEGER);
```

SQLGetEnvAttr

Summary

SQLGetEnvAttr returns the current setting of an environment attribute.

Syntax

```
SQLRETURN SQLGetEnvAttr( SQLHENV           henv
                        SQLINTEGER        Attribute,
                        SQLPOINTER       ValuePtr,
                        SQLINTEGER        BufferLength,
                        SQLINTEGER *      StringLengthPtr);
```

Arguments

henv [Input]: Environment handle.

Attribute[Input]: Attribute to retrieve.

ValuePtr[Output]: Pointer to a buffer in which to return the current value of the attribute specified by *Attribute*.

BufferLength [Input]: The value of this parameter depends upon following conditions,

- If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*.
- If **ValuePtr* is an integer, *BufferLength* is ignored.
- If **ValuePtr* is a Unicode string (when calling **SQLGetEnvAttrW**), the *BufferLength* argument must be an even number.
- If the attribute value is not a character string, *BufferLength* is unused.

StringLengthPtr[Output]: A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*.

- If *ValuePtr* is a null pointer, no length is returned.
- If the attribute value is a character string and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or
SQL_INVALID_HANDLE.

Example :

```
SQLRETURN retCode = SQLGetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, &value, 0,
&buffer);
```

SQLGetConnectAttr

Summary

SQLGetConnectAttr returns the current setting of a connection attribute.

Syntax

```
SQLRETURN SQLGetConnectAttr( SQLHDBC      hdbc,
                             SQLINTEGER   Attribute,
                             SQLPOINTER   ValuePtr,
                             SQLINTEGER   BufferLength,
                             SQLINTEGER *  StringLengthPtr);
```

Arguments

hdbc [Input]: Connection handle.

Attribute [Input]: Attribute to retrieve.

ValuePtr [Output]: Pointer to a buffer in which to return the current value of the attribute specified by *Attribute*.

BufferLength [Input]: The value of this parameter depends upon following conditions,

- If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*.
- If **ValuePtr* is an integer, *BufferLength* is ignored.
- If **ValuePtr* is a Unicode string (when calling **SQLGetEnvAttrW**), the *BufferLength* argument must be an even number.
- If the attribute value is not a character string, *BufferLength* is unused.

StringLengthPtr[Output]: A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*.

- If *ValuePtr* is a null pointer, no length is returned.
- If the attribute value is a character string and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLRETURN retCode = SQLGetConnectAttr (hdbc, SQL_ATTR_AUTOCOMMIT, &value,
                                       SQL_IS_INTEGER,&buffer);
```

SQLGetStmtAttr

Summary

SQLGetStmtAttr returns the current setting of a statement attribute.

Syntax

```
SQLRETURN SQLGetStmtAttr( SQLHSTMT  hstmt,  
                          SQLINTEGER  Attribute,  
                          SQLPOINTER  ValuePtr,  
                          SQLINTEGER  BufferLength,  
                          SQLINTEGER * StringLengthPtr);
```

Arguments

hstmt [Input]: Statement handle.

Attribute [Input]: Attribute to retrieve.

ValuePtr [Output]: Pointer to a buffer in which to return the current value of the attribute specified by

Attribute.

BufferLength [Input]: The value of this parameter depends upon following conditions,

- If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*.
- If **ValuePtr* is an integer, *BufferLength* is ignored.
- If **ValuePtr* is a Unicode string (when calling **SQLGetEnvAttrW**), the *BufferLength* argument must be an even number.
- If the attribute value is not a character string, *BufferLength* is unused.

StringLengthPtr[Output]: A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*.

- If *ValuePtr* is a null pointer, no length is returned.
- If the attribute value is a character string and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or
SQL_INVALID_HANDLE.

Example :

```
retCode = SQLGetStmtAttr (hstmt, SQL_ATTR_CURSOR_TYPE, &value, SQL_IS_INTEGER,  
&buffer);
```

SQLPrepare

Summary

SQLPrepare prepares an SQL statement later execution.

Syntax

```
SQLRETURN SQLPrepare(   SQLHSTMT   StatementHandle,  
                        SQLCHAR *   StatementText,  
                        SQLINTEGER   TextLength);
```

Arguments

StatementHandle[Input]: Statement handle.

StatementText[Input]: SQL text string.

TextLength[Input]: Length of *StatementText.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLRETURN retCode;  
SQLINTEGER buffer;value=10;  
retCode = SQLPrepare(hstmt,(SQLCHAR*) "select * from post where postID = ? ",SQL_NTS);  
retCode =  
SQLBindParameter(hstmt,1,1,SQL_C_SLONG,SQL_INTEGER,10,0,&value,sizeof(SQLINTEGE  
R), &buffer);  
retCode = SQLExecute(hstmt);
```

SQLNumParams

Summary

This function returns the number of parameters in an SQL statement.

Syntax

```
SQLRETURN SQL_API SQLNumParams(SQLHSTMT hstmt, SQLSMALLINT  
*param_count_ptr);
```

Arguments

hstmt: Statement handle

param_count_ptr [Output]: Pointer to a buffer in which number of parameters in the statement are returned.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLSMALLINT colCount;  
SQLRETURN retCode = SQLNumParams(hstmt,&colCount);
```

SQLBindParameter

Summary

This API binds a buffer to a parameter marker in an SQL statement , that is later used at execution time.

Syntax

```
SQLRETURN SQL_API SQLBindParameter(SQLHSTMT hstmt,
                                   SQLUSMALLINT ipar, SQLSMALLINT fParamType,
                                   SQLSMALLINT fCType,
                                   SQLSMALLINT fSqlType, SQLUINTEGER
                                   cbColDef, SQLSMALLINT ibScale, SQLPOINTER
                                   rgbValue, SQLINTEGER cbValueMax, SQLINTEGER
                                   *pcbValue);
```

Arguments

hstmt : Statement handle.

ipar [Input]: Parameter number in sequential increasing order, starting at 1.

fParamType [Input]: IN , OUT or IN OUT

fCtype [Input]: The C data type of the parameter.

fSqlType [Input]: The SQL data type of the parameter which matches with the sql type.

cbColDef [Input]: The column size of corresponding parameter marker.

ibScale [Input]: The decimal digits of column or expression of corresponding parameter marker.

rgbValue [Input]: A pointer to a buffer, which contains the parameter data to be supplied at run time .

cbValueMax [Input]: Length of the *ParameterValuePtr* buffer in bytes.

pcbValue [Input]: A pointer to a buffer for the parameter's length.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example :

```
SQLINTEGER buffer;value=10;
SQLRETURN retCode =
SQLBindParameter(hstmt,1,1,SQL_C_SLONG,SQL_INTEGER,10,0,&value,sizeof(SQLINTEGE
R), &buffer);
```

SQLExecute

Summary

This function executes a prepared statement, with the help of the current values of the parameter marker variables (if any parameter marker exists).

Syntax

```
SQLRETURN SQL_API SQLExecute(SQLHSTMT hstmt);
```

Arguments

hstmt: Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, or SQL_INVALID_HANDLE

Example :

```
SQLRETURN retCode;
SQLINTEGER buffer,value=10;
retCode = SQLPrepare(hstmt,(SQLCHAR*) "select * from post where postID = ? ",SQL_NTS);
retCode =
SQLBindParameter(hstmt,1,1,SQL_C_SLONG,SQL_INTEGER,10,0,&value,sizeof(SQLINTEGE
R), &buffer);
retCode = SQLExecute(hstmt);
```

SQLExecDirect

Summary

This function is used to prepare and execute a SQL statement once.

Syntax

```
SQLRETURN SQL_API SQLExecDirect(SQLHSTMT hstmt,SQLCHAR *szSqlStr,SQLINTEGER  
cbSqlStr);
```

Arguments

hstmt: Statement handle.

szSqlStr [Input]: SQL statement to be executed.

cbSqlStr [Input]:Length of szSQLStr in bytes or SQL_NTS.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO ,SQL_ERROR, SQL_NO_DATA, or
SQL_INVALID_HANDLE

Example :

```
SQLRETURN retCode = SQLExecute(hstmt,(SQLCHAR* ) " select * from post",SQL_NTS);
```

SQLNativeSql

Summary

SQLNativeSql returns the SQL string as modified by the driver. It does not execute the SQL statement.

Syntax

```
SQLRETURN SQL_API SQLNativeSql( SQLHDBC hdbc, SQLCHAR *szSqlStrIn,  
                                SQLINTEGER cbSqlStrIn,SQLCHAR *szSqlStr,  
                                SQLINTEGER cbSqlStrMax, SQLINTEGER *pcbSqlStr)  
;
```

Arguments

hdbc: Connection handle.

szSqlStrIn [Input]: SQL text string to be translated in SQL format.

cbSqlStrIn [Input]: Length of string to be translated in SQL format in bytes or SQL_NTS.

szSqlStr [Output]: Pointer to a buffer in which translated SQL string is returned.

cbSqlStrMax [Input]: Length of the * **szSqlStr** buffer.

pcbSqlStr [Output]: Pointer to a buffer which contains the returned the total number of bytes available to return in * **szSqlStr**.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR outputQuery[1024];  
SQLINTEGER outputlen ;  
SQLRETURN retCode = SQLNativeSql(hdbc,(SQLCHAR*)"select * from post where post ID =  
12} ", SQL_NTS,(SQLCHAR *)OutputQuery,1024,&outputlen);
```

SQLDescribeParam

Summary

This function returns the description of a parameter marker associated with a prepared SQL statement.

Syntax

```
SQLRETURN SQL_API SQLDescribeParam(    SQLHSTMT hstmt,
                                       SQLUSMALLINT ipar,
                                       SQLSMALLINT *pfSqlType,
                                       SQLINTEGER *pcbColDef,
                                       SQLSMALLINT *pibScale,
                                       SQLSMALLINT *pfNullable);
```

Arguments

Hstmt: Statement handle.

ipar [input]: Contains the value for N-th Parameter in prepared statement. Parameter order starts at 1

pfSqlType [Output]: Pointer to a buffer which returns the SQL data type of parameter.

pcbColDef [Output]: Pointer to a buffer which returns the size of the column.

pibScale [Output]: Pointer to a buffer which returns the scale of the column.

pfNullable [Output]: Pointer to a buffer which indicates that whether this column shall allow null values or not. The allowed values are,

- SQL_NO_NULLS: NULL values are not allowed (default).
- SQL_NULLABLE: NULL values are allowed.
- SQL_NULLABLE_UNKNOWN: The driver cannot determine if the parameter allows NULL values.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLSMALLINT sqltype, ibscale, nullable, retCode ;
SQLINTEGER coldef;
retCode = SQLPrepare(hstmt, (SQLCHAR*)" insert into post values (?, ?, ?) ", SQL_NTS);
retCode = SQLDescribeParam(hstmt, 1, &sqltype, &coldef, &ibscale, &nullable);
```

SQLRowCount

Summary

This function returns the number of rows affected by an **UPDATE**, **INSERT**, or **DELETE** statement;

Syntax

```
SQLRETURN SQL_API SQLRowCount (SQLHSTMT hstmt,  SQLINTEGER *rowCountPtr);
```

Arguments

hstmt: Statement handle

rowCountPtr [Output]: Points to a buffer in which the row count is returned.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLINTEGER rowcountPtr;  
SQLRETURN retCode ;  
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"insert into post values(12,18,'V.C')",SQL_NTS);  
retCode = SQLRowCount(hstmt,&rowcountPtr);
```

SQLNumResultCols

Summary

This function returns the number of columns from previously executed DQL statement in statement handle.

Syntax

```
SQLRETURN SQL_API SQLNumResultCols(SQLHSTMT hstmt, SQLSMALLINT *no_of_col);
```

Arguments

hstmt : Statement handle

no_of_col[Output]: Number of columns in the result set.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLSMALLINT colCount;  
SQLRETURN retCode ;  
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select * from post", SQL_NTS );  
retCode = SQLNumResultCols (hstmt,& colCount);
```

SQLDescribeCol

Summary

This function returns the result descriptor—column name, type, column size, decimal digits, and null ability—for a column in the result set on every call.

Syntax

```
SQLRETURN SQL_API SQLDescribeCol ( SQLHSTMT hstmt,
                                   SQLUSMALLINT ColumnNumber,
                                   SQLCHAR *ColumnName,
                                   SQLSMALLINT BufferLength,
                                   SQLSMALLINT *NameLengthPtr,
                                   SQLSMALLINT *DataTypePtr,
                                   SQLINTEGER *ColumnSizePtr,
                                   SQLSMALLINT *DecimalDigitsPtr,
                                   SQLSMALLINT *NullablePtr );
```

Arguments

Hstmt: Statement handle

ColumnNumber [Input]: Column number of result data, ordered sequentially in increasing column order, starting at 1 (1 based indexing).

ColumnName [Output]: Pointer to a buffer in which column name is returned.

BufferLength[Input]: Length of the *ColumnName* buffer, in characters.

NameLengthPtr [Output]: Pointer to a buffer in which the total number of bytes available in **ColumnName buffer* is returned.

DataTypePtr[Output]: Pointer to a buffer in which the SQL data type of the column is returned.

ColumnSizePtr [Output]: Pointer to a buffer which contains the size of the column in the data source.

DecimalDigitsPtr [Output]: Pointer to a buffer which contains the number of decimal digits of the column in the data source. If the number of decimal digits cannot be determined, the driver returns 0.

NullablePtr [Output]: Pointer to a buffer which indicates that whether this column shall allow null values or not. The allowed values are,

- SQL_NO_NULLS: NULL values are not allowed (default).
- SQL_NULLABLE: NULL values are allowed.
- SQL_NULLABLE_UNKNOWN: The driver cannot determine if the parameter allows NULL values.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR * colname = new SQLCHAR[129];
SQLSMALLINT totalbyte;
SQLSMALLINT datatype;
SQLINTEGER colsize;
SQLSMALLINT decdigit;
SQLSMALLINT nullable;
SQLRETURN retCode;
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select * from post", SQL_NTS );
retCode = SQLDescribeCol(hstmt,1,colname,128,&totalbyte,& datatype, &colsize,
&decdigit,&nullable);
```

SQLColAttribute

Summary

This function returns descriptor information for a column in a result set. Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

Syntax

```
SQLRETURN SQL_API SQLColAttribute(SQLHSTMT hstmt,  
                                  SQLUSMALLINT ColumnNumber,  
                                  SQLUSMALLINT FieldIdentifier,  
                                  SQLPOINTER CharacterAttributePtr,  
                                  SQLSMALLINT BufferLength,  
                                  SQLSMALLINT StringLengthPtr,  
                                  SQLPOINTER NumericAttributePtr);
```

Arguments

hstmt: Statement handle.

ColumnNumber [Input]: Column number in the result set for which you want to get descriptor information.

FieldIdentifier [Input]: The field in row / *ColumnNumber*.

CharacterAttributePtr [Output]: Pointer to a buffer in which the value in the FieldIdentifier field is returned, otherwise, the field is unused.

BufferLength [Input]: Length of (buffer) CharacterAttributePtr in bytes.

StringLengthPtr [Output]: Pointer to a buffer which contains the total number of bytes available to return.

NumericAttributePtr[Output]: Pointer to an integer buffer .

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR * schemaName = new SQLCHAR[129];  
SQLRETURN retCode;  
SQLSMALLINT strlenptr=0;  
SQLPOINTER numattrlenptr  
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select * from post", SQL_NTS );  
retCode = SQLColAttribute(hstmt,1,SQL_DESC_SCHEMA_NAME,  
schemaName,128, &strlenptr,&numattrlenptr);
```

SQLBindCol

Summary

This function binds application data buffers to columns in the result set.

Syntax

```
SQLRETURN SQL_API SQLBindCol( SQLHSTMT hstmt,  
                               SQLUSMALLINT icol,  
                               SQLSMALLINT fCType,  
                               SQLPOINTER rgbValue,  
                               SQLINTEGER cbValueMax,  
                               SQLINTEGER *pcbValue);
```

Arguments

hstmt: Statement handle.

icol[Input]: Number of the result set column to bind.

fCType[Input]: C compatible data type of this column.

rgbValue[Input]: Pointer to the data buffer for binding the column. Data is returned at fetch time in this buffer.

cbValueMax [Input]: Length of the * rgbValue buffer in bytes.

pcbValue: Pointer to the length/indicator buffer to bind to the column.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR * postName = new SQLCHAR[129];  
SQLRETURN retCode;  
SQLINTEGER cbbuffer;  
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select postName from post", SQL_NTS );  
retCode = SQLBindCol(hstmt,1 , SQL_C_CHAR,postName,128, &cbbuffer);
```

SQLFetch

Summary

This method fetches the next rowset of data from the result set and returns data for all bound columns .

Syntax

```
SQLRETURN SQLFetch( SQLHSTMT hstmt);
```

Arguments

hstmt : Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR * postName = new SQLCHAR[129];
SQLRETURN retCode;
SQLINTEGER cbbuffer;
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select postName from post", SQL_NTS );
retCode = SQLBindCol(hstmt,1 , SQL_C_CHAR,postName,128, &cbbuffer);
retCode = SQLFetch(hstmt);
while(retCode == SQL_SUCCESS || retCode == SQL_SUCCESS_WITH_INFO )
/*Process data here */
retCode = SQLFetch(hstmt);
if(retCode == SQL_NO_DATA)
break; // Data is not available or end of result set found.
}
```

SQLExtendedFetch

Summary

SQLExtendedFetch fetches the specified rowset of data from the result set and returns data for all bound columns. Rowsets can be specified at an absolute or relative position.

Syntax

```
SQLRETURN SQLExtendedFetch( SQLHSTMT      hstmt,  
                             SQLUSMALLINT  FetchOrientation,  
                             SQLINTEGER     FetchOffset,  
                             SQLUINTEGER *  RowCountPtr,  
                             SQLUSMALLINT * RowStatusArray);
```

Arguments

hstmt: Statement handle.

FetchOrientation [Input]: Type of fetch.

FetchOffset [Input]: Number of the row to fetch.

RowCountPtr [Output]: Pointer to a buffer in which to return the number of rows actually fetched.

RowStatusArray [Output]: Pointer to an array in which to return the status of each row.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLRETURN retCode = SQLExtendedFetch (hstmt, SQL_FETCH_NEXT, 0, &rowsFetched,  
rowStatus);
```

SQLFetchScroll

Summary

This function fetches the specified rowset of data from the result set and returns data for all bound columns. Rowsets can be specified at an absolute or relative position.

Syntax

```
SQLRETURN SQL_API SQLFetchScroll ( SQLHSTMT  hstmt,  
                                  SQLSMALLINT FetchOrientation,  
                                  SQLINTEGER  FetchOffset);
```

Arguments

hstmt: Statement handle.

***FetchOrientation* [Input]**: Type of fetch. Following fetch orientations are supported,

```
SQL_FETCH_NEXT  
SQL_FETCH_PRIOR  
SQL_FETCH_FIRST  
SQL_FETCH_LAST  
SQL_FETCH_ABSOLUTE  
SQL_FETCH_RELATIVE
```

***FetchOffset*[Input]**: Number of the row to fetch. The interpretation of this argument depends on the value of the *FetchOrientation* argument.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR * postName = new SQLCHAR[129];  
SQLRETURN retCode;  
SQLINTEGER cbbuffer;  
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select postName from post", SQL_NTS );  
retCode = SQLBindCol(hstmt,1 , SQL_C_CHAR,postName,128, &cbbuffer);  
//Fetch the next row  
retCode = SQLFetchScroll (hstmt, SQL_FETCH_NEXT,0);
```

SQLGetData

Summary

This method retrieves data for a single column in the result set. It can be called more times to retrieve variable-length data in parts. Before calling this method a application must call any of fetch type method .i.e. SQLFetch,SQLFetchScroll or SQLExtendedFetch.

Syntax

```
SQLRETURN SQL_API SQLGetData ( SQLHSTMT hstmt,
                               SQLUSMALLINT icol,
                               SQLSMALLINT fCType,
                               SQLPOINTER outputBufferPointer,
                               SQLINTEGER cbValueMax,
                               SQLINTEGER *outputBufferLengthPointer);
```

Arguments

hstmt: Statement handle.

ico/[Input]: Number of the result set column to bind.

fCType[Input]: C compatible data type of this column.

outputBufferPointer[Output]: Pointer to the buffer in which to return the data.

cbValueMax[Input]: Length of the * outputBufferPointer buffer in bytes.

outputBufferLengthPointer[Output]: Pointer to the buffer in which to return the length or indicator value. If this is a null pointer, no length or indicator value is returned.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLINTEGER buffer;
SQLCHAR *postName = new SQLCHAR[129];
retCode = SQLExecDirect(hstmt,(SQLCHAR*)"select postName from post", SQL_NTS );
retCode = SQLBindCol(hstmt,1 , SQL_C_CHAR,postName,128, &cbbuffer);
retCode = SQLFetch(hstmt);
retCode = SQLGetData(hstmt,1, SQL_C_CHAR,postName,128,&buffer);
// Process your data here .
```

SQLEndTran

Summary

This function requests a commit or rollback operation for all active operations on all statements associated with a connection. This function can also request that a commit or rollback operation be performed for all connections associated with an environment. `SQLTransact` is the alternative of this API function, DaffodilDB Driver also provide this function.

Syntax

```
SQLRETURN SQL_API SQLEndTran ( SQLSMALLINT HandleType,  
                               SQLHANDLE Handle,  
                               SQLSMALLINT CompletionType);
```

Argument

HandleType : Supported handle types are listed below

1. `SQL_HANDLE_ENV`
2. `SQL_HANDLE_DBC`
3. `SQL_HANDLE_STMT`

Handle : **[Input]**: The handle, of the type indicated by *HandleType*, indicating the scope of the transaction.

CompletionType **[Input]**: It must contain one of the following vales

1. `SQL_COMMIT`
2. `SQL_ROLLBACK`

Returns

`SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

Example

```
SQLRETURN retCode = SQLEndTran(SQL_HANDLE_DBC,hdbc,SQL_COMMIT);
```

SQLTables

Summary

SQLTables returns the list of table, catalog, or schema names, and table types, stored in a specific data source. The driver returns the information as a result set.

Syntax

```
SQLRETURN SQLTables( SQLHSTMT hstmt , SQLCHAR * CatalogName,
                    SQLSMALLINT NameLength1, SQLCHAR * SchemaName,
                    SQLSMALLINT NameLength2, SQLCHAR * TableName,
                    SQLSMALLINT NameLength3, SQLCHAR * TableType,
                    SQLSMALLINT NameLength4 );
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: Catalog name. The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

TableName [Input]: String search pattern for table names.

NameLength3 [Input]: Length in characters of *TableName.

TableType [Input]: List of table types to match.

NameLength4 [Input]: Length in characters of *TableType.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLTables returns a result set consisting of the columns Listed in Table A,

Column No.	Column Name
1	TABLE_CAT
2	TABLE_SCHEM
3	TABLE_NAME
4	TABLE_TYPE
5	REMARKS

Table A

Code Example

```

HSTMT hstmt;
UCHAR tableCatalog[255], tableSchma[255], tabName[255];
UCHAR tabType[255], remarks[255];
SQLINTEGER lenTableCatalog, lenTableSchma , lenTableType, lenRemarks, lenTableName
;

SQLRETURN retcode ;
//After allocating statement handle....
//Do necessary checking of value returned by SQLTables before processing further
retcode = SQLTables(hstmt ,
                    (UCHAR *)NULL, 0,          /* tab catalog
                    (UCHAR *)NULL, 0,          /* tab schema   */
                    (UCHAR *)" %", SQL_NTS,    /* table name   */
                    (UCHAR *)"TABLE", SQL_NTS); /* table type   */

/* Bind columns in result set to storage locations          */

SQLBindCol(hstmt, 1, SQL_C_CHAR, tableCatalog, 255, &lenTableCatalog);
SQLBindCol(hstmt, 2, SQL_C_CHAR, tableSchma, 255, &lenTableSchma);
SQLBindCol(hstmt, 3, SQL_C_CHAR, tabName, 255, &lenTableName);
SQLBindCol(hstmt, 4, SQL_C_CHAR, tabType, 255, &lenTableType);
SQLBindCol(hstmt, 5, SQL_C_CHAR, remarks, 255, &lenRemarks);
/* Now print out the records in the result set
retucode =
*/
while( retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
    printf("column 1 : Tabble catalog = %s\n", tableCatalog);
    printf("column 2 : Table schema = %s\n", tableSchma);
    printf("column 3 : Table name = %s\n", tabName);
    printf("column 4 : Table type = %s\n", tabType);
    printf("column 5 : Remarks = %s\n", remarks);
}

```

SQLTablePrivileges

Summary

SQLTablePrivileges returns a list of tables and the privileges associated with each table. The driver returns the information as a result set on the specified statement.

Syntax

```
SQLRETURN SQLTablePrivileges ( SQLHSTMT  hstmt , SQLCHAR *  CatalogName,
                               SQLSMALLINT  NameLength1, SQLCHAR *  SchemaName,
                               SQLSMALLINT  NameLength2, SQLCHAR *  TableName,
                               SQLSMALLINT  NameLength3, SQLCHAR *  ColumnName,
                               SQLSMALLINT  NameLength4);
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

TableName [Input]: String search pattern for table names.

NameLength3 [Input]: Length in characters of *TableName.

ColumnName [Input]: String search pattern for column names.

NameLength4 [Input]: Length in characters of *ColumnName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLTablePrivileges returns a result set consisting of the following columns

Code Example

```
SQLRETURN  retCode =
SQLTablePrivileges(hstmt,(SQLCHAR*)"catalogName",SQL_NTS,(S
QLCHAR*)"schemaName",SQL_NTS,(SQLCHAR*)"tableName",SQL
_NTS);
```

Column No.	Column Name	
1	TABLE_CAT	
2	TABLE_SCHEM	
3	TABLE_NAME	
4	GRANTOR	
5	GRANTEE	
6	PRIVILEGE	Table B
7	IS_GRANTABLE	

SQLColumns

Summary

SQLColumns returns the list of column names in specified tables. The driver returns this information as a result set on the specified statement handle.

Syntax

```
SQLRETURN SQLColumns (
    SQLHSTMT hstmt, SQLCHAR * CatalogName,
    SQLSMALLINT NameLength1, SQLCHAR *
    SchemaName,
    SQLSMALLINT NameLength2, SQLCHAR * TableName,
    SQLSMALLINT NameLength3, SQLCHAR *
    ColumnName,
    SQLSMALLINT NameLength4
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: Catalog name. The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

TableName [Input]: String search pattern for table names.

NameLength3 [Input]: Length in characters of *TableName.

ColumnName [Input]: String search pattern for column names.

NameLength4 [Input]: Length in characters of *ColumnName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLColumns returns a result set consisting of columns contained in table C.

```
rcode = SQLColumns(hstmt, NULL, 0, NULL, 0,
(SQLCHAR*)"POST", SQL_NTS, /* POST table */
NULL, 0); /* All columns */
```

Column No.	Column Name	
1	TABLE_CAT	
2	TABLE_SCHEM	
3	TABLE_NAME	
4	COLUMN_NAME	
5	DATA_TYPE	
6	TYPE_NAME	
7	COLUMN_SIZE	
8	BUFFER_LENGTH	
9	DECIMAL_DIGITS	
10	NUM_PREC_RADIX	
11	NULLABLE	
12	REMARKS	
13	COLUMN_DEF	
14	SQL_DATA_TYPE	
15	SQL_DATETIME_SUB	
16	CHAR_OCTET_LENGTH	
17	ORDINAL_POSITION	
18	IS_NULLABLE	

Table C

SQLColumnPrivileges

Summary

SQLColumnPrivileges returns a list of columns and associated privileges for the specified table. The driver returns the information as a result set on the specified StatementHandle.

Syntax

```
SQLRETURN SQLColumnPrivileges ( SQLHSTMT  hstmt , SQLCHAR *  CatalogName,
                                SQLSMALLINT NameLength1,
                                SQLCHAR *  SchemaName,
                                SQLSMALLINT  NameLength2,
                                SQLCHAR *  TableName,
                                SQLSMALLINT  NameLength3,
                                SQLCHAR *  ColumnName,
                                SQLSMALLINT  NameLength4 );
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: Catalog name. The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

TableName [Input]: String search pattern for table names.

NameLength3 [Input]: Length in characters of *TableName.

ColumnName [Input]: String search pattern for column names.

NameLength4 [Input]: Length in characters of *TableType.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

Code Example

```
SQLRETURN retCode =
SQLColumnPrivileges(hstmt,(SQLCHAR*)"catalogName",SQL_NTS,(SQLCHAR*)"schemaName
", SQL_NTS,(SQLCHAR*)"tableName", SQL_NTS,(SQLCHAR*)"columnName", SQL_NTS);
```

SQLColumnPrivileges returns a result set consisting of columns contained in table D.

Column No.	Column Name
1	TABLE_CAT
2	TABLE_SCHEM
3	TABLE_NAME
4	COLUMN_NAME
5	GRANTOR
6	GRANTEE
7	PRIVILEGE
8	IS_GRANTABLE

Table D

SQLPrimaryKeys

SQLPrimaryKeys returns the column names that make up the primary key for a table. The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.

Syntax

```
SQLRETURN SQLPrimaryKeys (SQLHSTMT hstmt ,
                          SQLCHAR * CatalogName,
                          SQLSMALLINT NameLength1,
                          SQLCHAR * SchemaName,
                          SQLSMALLINT NameLength2,
                          SQLCHAR * TableName,
                          SQLSMALLINT NameLength3,
                          SQLCHAR * ColumnName,
                          SQLSMALLINT NameLength4);
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: Catalog name. The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

TableName [Input]: String search pattern for table names.

NameLength3 [Input]: Length in characters of *TableName.

ColumnName [Input]: String search pattern for column names.

NameLength4 [Input]: Length in characters of *ColumnName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLPrimaryKeys returns a result set consisting of columns contained in table D.

Code Example

```
SQLRETURN retcode = SQLPrimaryKeys(hstmt, (SQLCHAR*)"catalogName", SQL_NTS,
                                   (SQLCHAR*)"schemaName", SQL_NTS, (SQLCHAR*)"tableName", SQL_NTS);
```

Column No.	Column Name
1	TABLE_CAT
2	TABLE_SCHEM
3	TABLE_NAME
4	COLUMN_NAME
5	KEY_SEQ
6	PK_NAME

Table E:

SQLForeignKeys

Summary

SQLForeignKeys can return:

- A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables).
- A list of foreign keys in other tables that refer to the primary key in the specified table.

The driver returns each list as a result set on the specified statement.

Syntax

```
SQLRETURN SQLForeignKeys( SQLHSTMT      stmt,
                          SQLCHAR *    PKCatalogName,
                          SQLSMALLINT  NameLength1,
                          SQLCHAR *    PKSchemaName,
                          SQLSMALLINT  NameLength2,
                          SQLCHAR *    PKTableName,
                          SQLSMALLINT  NameLength3,
                          SQLCHAR *    FKCatalogName,
                          SQLSMALLINT  NameLength4,
                          SQLCHAR *    FKSchemaName,
                          SQLSMALLINT  NameLength5,
                          SQLCHAR *    FKTableName,
                          SQLSMALLINT  NameLength6 );
```

Arguments

hstmt [Input]: Statement handle.

PKCatalogName[Input]: Primary key table catalog name.

NameLength1 [Input]: Length of * *PKCatalogName*.

PKSchemaName [Input]: Primary key table schema name.

NameLength2 [Input]: Length of * *PKSchemaName*, in bytes.

PKTableName[Input]: Primary key table name. *PKTableName* cannot contain a search pattern.

NameLength3 [Input]: Length of * *PKTableName*.

FKCatalogName [Input]: Foreign key table catalog name.

NameLength4 [Input]: Length of * *FKCatalogName*.

FKSchemaName[Input]: Foreign key table schema name.

NameLength5 [Input]: Length of * *FKSchemaName*.

FKTableName[Input]: Foreign key table name. *FKTableName* cannot contain a search pattern.

NameLength6 [Input]: Length of * FKTableName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLForeignKeys returns a result set consisting of the following columns :

Table F

Column Column Name

1	PKTABLE_CAT
2	PKTABLE_SCHEM
3	PKTABLE_NAME
4	PKCOLUMN_NAME
5	FKTABLE_CAT
6	FKTABLE_SCHEM
7	FKTABLE_NAME
8	FKCOLUMN_NAME
9	KEY_SEQ
10	UPDATE_RULE
11	DELETE_RULE
12	PK_NAME
13	FK_NAME
14	DEFERRABILITY

Code Example

```
retcode = SQLForeignKeys(hstmt,
                          NULL, 0,
                          NULL, 0,
                          (SQLCHAR *)"POST",
                          SQL_NTS,
                          NULL, 0,
                          NULL, 0,
                          NULL, 0,
                          NULL, 0);
```

SQLGetTypeInfo

Summary

SQLGetTypeInfo returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set. The data types are intended for use in Data Definition Language (DDL) statements.

Syntax

```
SQLRETURN SQLGetTypeInfo( SQLHSTMT hstmt, SQLSMALLINT DataType);
```

Arguments

hstmt [Input]: Statement handle.

DataType[Input]: The SQL data type. This must be one of the values in the SQL Data Types section in **Table 2** (Chapter 1).

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLGetTypeInfo returns a result set consisting of the following columns:

Column No.	Column Name
1	TYPE_NAME
2	DATA_TYPE
3	COLUMN_SIZE
4	LITERAL_PREFIX
5	LITERAL_SUFFIX
6	CREATE_PARAMS
7	NULLABLE
8	CASE_SENSITIVE
9	SEARCHABLE
10	UNSIGNED_ATTRIBUTE
11	FIXED_PREC_SCALE
12	AUTO_UNIQUE_VALUE
13	LOCAL_TYPE_NAME
14	MINIMUM_SCALE
15	MAXIMUM_SCALE
16	SQL_DATA_TYPE
17	SQL_DATETIME_SUB
18	NUM_PREC_RADIX
19	INTERVAL_PRECISION

Table G

Code Example

```
SQLRETURN retCode = SQLGetTypeInfo(hstmt , /*Statement handle*/ SQL_ALL_TYPES )  
;/*SQL data type or SQL_ALL_TYPES for all data types*/
```

SQLProcedures

Summary

SQLProcedures returns the list of procedure names stored in a specific data source. Procedure is a generic term used to describe an executable object, or a named entity that can be invoked using input and output parameters.

Syntax

```
SQLRETURN SQLProcedures ( SQLHSTMT  hstmt,  
                          SQLCHAR *  CatalogName,  
                          SQLSMALLINT NameLength1,  
                          SQLCHAR *  SchemaName,  
                          SQLSMALLINT NameLength2,  
                          SQLCHAR *  ProcName,  
                          SQLSMALLINT NameLength3);
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: Catalog name. The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

ProcName [Input]: String search pattern for procedure names.

NameLength3 [Input]: Length in characters of *ProcName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLProcedures returns a result set consisting of the following columns:

Column No.	Column Name
1	PROCEDURE_CAT
2	PROCEDURE_SCHEM
3	PROCEDURE_NAME
4	NUM_INPUT_PARAMS
5	NUM_OUTPUT_PARAMS
6	NUM_RESULT_SETS
7	REMARKS
8	PROCEDURE_TYPE

Table 5: Columns returned in SQLProcedures result set

Code Example

```
retCode = SQLProcedures(hstmt,(SQLCHAR*)"catalogName",SQL_NTS,  
(SQLCHAR*)"catalogName",SQL_NTS,(SQLCHAR*)"procedureName",SQL_NTS);
```

SQLProcedureColumns

Summary

SQLProcedureColumns returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. The driver returns the information as a result set on the specified statement.

Syntax

```
SQLRETURN SQLProcedureColumns( SQLHSTMT  StatementHandle,  
                               SQLCHAR *  CatalogName,  
                               SQLSMALLINT NameLength1,  
                               SQLCHAR *  SchemaName,  
                               SQLSMALLINT NameLength2,  
                               SQLCHAR *  ProcName,  
                               SQLSMALLINT NameLength3,  
                               SQLCHAR *  ColumnName,  
                               SQLSMALLINT NameLength4);
```

Arguments

hstmt [Input]: Statement handle.

CatalogName [Input]: Catalog name. The CatalogName argument accepts search patterns.

NameLength1 [Input]: Length in characters of *CatalogName.

SchemaName [Input]: String search pattern for schema names.

NameLength2 [Input]: Length in characters of *SchemaName.

ProcName [Input]: String search pattern for table names.

NameLength3 [Input]: Length in characters of *ProcName.

ColumnName [Input]: String search pattern for column names.

NameLength4 [Input]: Length in characters of *ColumnName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

SQLProcedureColumns returns a result set consisting of the following columns:

Column No.	Column Name
1	PROCEDURE_CAT
2	PROCEDURE_SCHEM
3	PROCEDURE_NAME
4	COLUMN_NAME
5	COLUMN_TYPE
6	DATA_TYPE
7	TYPE_NAME
8	PRECISION
9	LENGTH
10	SCALE
11	RADIX
12	NULLABLE
13	REMARKS

Code Example

```
SQLRETURN retCode=  
SQLProcedureColumns(hstmt,(SQLCHAR*)"catalogName",SQL_NTS,(SQLCHAR*)"schemaName",SQL_NTS,(SQLCHAR*)"procedureName",SQL_NTS,(SQLCHAR*)"columnName",SQL_NTS);
```

SQLGetDiagRec

Summary

This function returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information.

Syntax

```
SQLRETURN SQL_API SQLGetDiagRec (SQLSMALLINT HandleType,  
                                SQLHANDLE Handle,  
                                SQLSMALLINT RecNumber,  
                                SQLCHAR *Sqlstate,  
                                SQLINTEGER *NativeErrorPtr,  
                                SQLCHAR *MessageText,  
                                SQLSMALLINT BufferLength,  
                                SQLSMALLINT *TextLengthPtr)
```

Arguments

HandleType: Supported handle types are,

1. SQL_HANDLE_ENV
2. SQL_HANDLE_DBC
3. SQL_HANDLE_STMT

Handle: Specified handle, that may be environmental, connection or statement handle.

RecNumber: For DaffodilDB Driver, it should always be 1.

Sqlstate[Output]: Pointer to a buffer in which to return a five-character SQLSTATE code pertaining to the diagnostic record *RecNumber*.

NativeErrorPtr[Output]: Pointer to a buffer in which to return the native error code, specific to the data source.

MessageText [Output]: Pointer to a buffer in which to return the diagnostic message text string.

BufferLength[Input]: Length (in bytes) of the **MessageText* buffer.

TextLengthPtr[Output]: Pointer to a buffer in which to return the total number of bytes available to return in **MessageText* buffer.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLCHAR state[128];
SQLCHAR message[128];
SQLINTEGER nativeError;
SQLSMALLINT o_p_length_of_message;
SQLSMALLINT retCode;
retCode = SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, state, &nativeError, message, 128,
                        &o_p_length_of_message);
```

SQLCancel

Summary

This function cancels the processing on a statement.

Syntax

```
SQLRETURN SQLCancel( SQLHSTMT hstmt);
```

Arguments

hstmt: Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Example

```
SQLRETURN retCode = SQLCancel(hstmt);
```

Appendix A - Diagnostics: ODBC SQL States and their Description

Following Table lists error codes and the messages corresponding to them.

Error Code	Error Message
01000	General warning
01004	String data, right truncated
01S02	Option value changed
07005	Prepared statement not a cursor-specification
07006	Restricted data type attribute violation
07009	Invalid descriptor index
08002	Connection name in use
08003	Connection does not exist
24000	Invalid cursor state
25000	Invalid transaction state
25S01	Transaction state unknown
HY000	General driver defined error.
HY001	Memory allocation error
HY002	Invalid column number
HY003	Invalid application buffer type
HY004	Invalid SQL data type
HY009	Invalid use of null pointer
HY010	Function sequence error
HY011	Attribute can not be set now
HY024	Invalid attribute value
HY090	Invalid string or buffer length
HY093	Invalid parameter number
HY106	Fetch type out of range
HYC00	Optional feature not implemented
IM001	Driver does not support this function
IM002	Data source name not found and no default driver specified
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed
IM006	Driver's SQLSetConnectAttr failed
IM007	No data source or driver specified; dialog prohibited
IM008	Dialog failed
IM010	Data source name too long