REMSTAR ODBC CLASS LIBRARY

USER'S GUIDE

wxDB CLASS **5**

CONSTRUCTOR 5
  wxDB::wxDB *5*
PUBLIC MEMBER FUNCTIONS 6
  wxDB::Catalog() *6*
  wxDB::Close() *6*
  wxDB::CommitTrans() *7*
  wxDB::CreateView() *7*
  wxDB:: DispAllErrors() *8*
  wxDB::DispNextError() *8*
  wxDB::ExecSql() *9*
  wxDB::GetColumns() *9*
  wxDB::GetDatabaseName() *10*
  wxDB::GetDataSource() *10*
  wxDB::GetHDBC() *10*
  wxDB::GetHENV() *10*
  wxDB::GetHSTMT() *11*
  wxDB::GetNextError() *11*
  wxDB::GetPassword() *11*
  wxDB::GetUsername() *11*
  wxDB::Grant() *12*
  wxDB::IsOpen() *12*
  wxDB::Open() *12*
  wxDB::RollbackTrans() *13*
  wxDB::SqlLog() *13*
  wxDB::TableExists() *13*
  wxDB::TranslateSqlState() *14*
  wxDB::WriteSqlLog() *14*
PUBLIC MEMBER VARIABLES 15
  wxDB::cbErrorMsg *15*
  wxDB::dbInf *15*
  wxDB::DB_STATUS *16*
  wxDB::errorList *17*
  wxDB::errorMsg *18*
  wxDB::hdbc *18*
  wxDB::henv *18*
  wxDB::hstmt *18*
  wxDB::nativeError *19*
  wxDB::silent *19*
  wxDB::sqlState *19*
  wxDB::typeInfDate *19*
  wxDB::typeInfFloat *19*
  wxDB::typeInfInteger *20*
  wxDB::typeInfVarchar *20*

wxTable CLASS **21**

CONSTRUCTORS/DESTRUCTORS 21
  wxTable::wxTable() *21*
  wxTable::~wxTable() *23*
PUBLIC MEMBER FUNCTIONS 24
  wxTable::CanSelectForUpdate() *24*
  wxTable::CanUpdByROWID() *24*
  wxTable::ClearMemberVars() *25*

wxTable::CloseCursor() *25*
wxTable::Count() *25*
wxTable::CreateIndex() *26*
wxTable::CreateTable() *26*
wxTable::DB_STATUS() *27*
wxTable::Delete() *27*
wxTable::DeleteMatching() *27*
wxTable::DeleteWhere() *28*
wxTable::GetCursor() *29*
wxTable::GetDeleteStmt() *29*
wxTable::GetFirst() *29*
wxTable::GetLast() *29*
wxTable::GetNext() *30*
wxTable::GetPrev() *30*
wxTable::GetRowNum() *30*
wxTable::GetSelectStmt() *30*
wxTable::GetUpdateStmt() *31*
wxTable::GetWhereClause() *31*
wxTable::Insert() *32*
wxTable::IsColNull() *32*
wxTable::IsCursorClosedOnCommit() *33*
wxTable::Open() *33*
wxTable::operator++ *33*
wxTable::operator-- *34*
wxTable::Query() *34*
wxTable::QueryBySqlStmt() *35*
wxTable::QueryMatching() *36*
wxTable::QueryOnKeyFields() *37*
wxTable::Refresh() *38*
wxTable::SetColDefs() *38*
wxTable::SetCursor() *39*
wxTable::SetQueryTimeout() *39*
wxTable::Update() *40*
wxTable::Update() *40*
wxTable::UpdateWhere() *41*
PUBLIC MEMBER VARIABLES 41
wxTable::c0 .... *41*
wxTable::c1 .... *42*
wxTable::c2 .... *42*
wxTable::colDefs *42*
wxTable::from *42*
wxTable::hdbc *43*
wxTable::henv *43*
wxTable::hstmt *43*
wxTable::hstmtDelete *43*
wxTable::hstmtInsert *43*
wxTable::hstmtUpdate *44*
wxTable::noCols *44*
wxTable::orderBy *44*
wxTable::pDb *44*
wxTable::selectForUpdate *45*
wxTable::tableName *45*
wxTable::where *45*

CCOLDEF **46**

OVERVIEW 46
CLASS DECLARATION 46
CLASS DETAILS 46

CIDXDEF 48

OVERVIEW 48
CLASS DECLARATION 48
CLASS DETAILS 48

CCOLINF 49

OVERVIEW 49
CLASS DECLARATION 49

MISCELLANEOUS FUNCTIONS 50

CloseDbConnections() 50
FreeDbConnection() 50
GetDataSource() 50
GetDbConnection() 51

APPENDIX A – ODBC RESERVED WORDS 52

LIST OF RESERVED WORDS 52

wxDB Class

Constructor

wxDB::wxDB

Declaration

wxDB(HENV &aHenv)

Arguments

aHenv	A variable of type HENV (environment handle).

Comments

This is the constructor for the wxDB class. The wxDB object must be created and opened before any database activity can occur.

Example

```
wxDB sampleDB(Db.Henv);
if (! sampleDB.Open(Db.Dsn, Db.Uid, Db.AuthStr))
{
   // Error opening data source
```

Public Member Functions

wxDB::Catalog()

Declaration

Bool Catalog(char *userID, char *fileName="Catalog.txt")

Arguments

userID	Database user name to use in accessing the database. All tables to which this user has rights will be evaluated in the catalog.
fileName	OPTIONAL argument. Name of the text file to create and write the DB catalog to.

Comments

Allows a data “dictionary” of the data source to be created, dumping pertinent information about all data tables to which the user specified in userID has access.

Example

```
=========================== =========================== ===================== ========= =========
TABLE NAME                  COLUMN NAME                 DATA TYPE             PRECISION    LENGTH
=========================== =========================== ===================== ========= =========
EMPLOYEE                    RECID                       (0008)NUMBER                 15         8
EMPLOYEE                    USER_ID                     (0012)VARCHAR2               13        13
EMPLOYEE                    FULL_NAME                   (0012)VARCHAR2               26        26
EMPLOYEE                    PASSWORD                    (0012)VARCHAR2               26        26
EMPLOYEE                    START_DATE                  (0011)DATE                   19        16
```

wxDB::Close()

Declaration

void Close(void)

Comments

At the end of your program, when you have finished all of your database work, you must close the ODBC connection to the data source. There are actually four steps involved in doing this as illustrated in the example.

Example

```
// Commit any open transactions on the data source
sampleDB.CommitTrans();

// Delete any remaining wxTable objects allocated with new
delete parts;

// Close the wxDB connection when finished with it
sampleDB.Close();

// Free Environment Handle that ODBC uses
if (SQLFreeEnv(Db.Henv) != SQL_SUCCESS)
```

```
{
  // Error freeing environment handle
}
```

wxDB::CommitTrans()

Declaration

Bool CommitTrans(void)

Comments

Call this member function to permanently "commit" changes to the database. Transactions begin implicitly as soon as you make a change to the database. At any time thereafter, you can save your work to the database ("Commit") or roll back all of your changes ("Rollback"). Calling this member function commits **all** open transactions on this ODBC connection.

Special Note re: Cursors

It's important to understand that different database/ODBC driver combinations handle transactions differently. One thing in particular that you must pay attention to is cursors, in regard to transactions. Cursors are what allow you to scroll through records forward and backward and to manipulate records as you scroll through them. When you issue a query, a cursor is created behind the scenes. The cursor keeps track of the query and keeps track of the current record pointer. *After you commit or rollback a transaction, the cursor may be closed automatically.* This means you must requery the data source before you can perform any additional work against the wxTable object. This is only necessary however if the data source closes the cursor after a commit or rollback. Use the wxTable::IsCursorClosedOnCommit() member function to determine the data source's transaction behavior. Note, it would be very inefficient to just assume the data source closes the cursor and always requery. This could put a significant, unnecessary load on data sources that leave the cursors open after a transaction.

wxDB::CreateView()

Declaration

Bool CreateView(char *viewName, char *colList, char *pSqlStmt)

Arguments

viewName	The name of the view. E.g. PARTS_V
colList	OPTIONAL argument. Pass in a comma delimited list of column names if you wish to explicitly name each column in the result set. If not, pass in an empty string.
pSqlStmt	Pointer to the select statement portion of the CREATE VIEW statement. Must be a complete, valid SQL SELECT statement.

Comments

CreateView() does just what it's name implies, creates a SQL VIEW. A view is a logical table that derives columns from one or more other tables or views. Once the view is created, it can be queried exactly like any other table in the database.

Example

```
// Incomplete code sample
db.CreateView(“PARTS_SD1”, “PN, PD, QTY”,  “SELECT PART_NO, PART_DESC,
QTY_ON_HAND * 1.1 FROM PARTS WHERE STORAGE_DEVICE = 1”);

// PARTS_SD1 can now be queried just as if it were a data table.
// e.g. SELECT PN, PD, QTY FROM PARTS_SD1
```

wxDB:: DispAllErrors()

Declaration

```
Bool DispAllErrors(HENV aHenv,
                   HDBC aHdbc = SQL_NULL_HDBC,
                   HSTMT aHstmt = SQL_NULL_HSTMT)
```

Arguments

aHenv	A handle to the ODBC environment.
aHdbc	A handle to the ODBC connection. Pass this in if the ODBC function call that erred out required a hdbc or hstmt argument.
AHstmt	A handle to the ODBC statement being executed against. Pass this in if the ODBC function call that erred out required a hstmt argument.

Comments

This member function will display all of the ODBC error messages for the last ODBC function call that was made. Normally used internally within the ODBC class library. Would be used externally if calling ODBC functions directly (i.e. SQLFreeEnv()).

Example

```
if (SQLExecDirect(hstmt, (UCHAR FAR *) pSqlStmt, SQL_NTS) != SQL_SUCCESS)
  return(db.DispAllErrors(db.henv, db.hdbc, hstmt));   // Display all ODBC errors for this stmt
```

wxDB::DispNextError()

Declaration

```
void DispNextError(void)
```

Comments

This function is normally used internally within the ODBC class library. It could be used externally if calling ODBC functions directly. This function works in conjunction with GetNextError() when errors (or sometimes informational messages) returned from ODBC need to be analyzed rather than simply displaying them as an error. GetNextError() retrieves the next ODBC error from the ODBC error queue. The wxDB member variables “sqlState”, “nativeError” and “errorMsg” could then be evaluated. To display the error retrieved, DispNextError() could then be called. The combination of GetNextError() and DispNextError() can be used to iteratively step through the errors returned from ODBC evaluating each one in context and displaying the ones you chose to.

Example

```
// Drop the table before attempting to create it
```

```
sprintf(sqlStmt, "DROP TABLE %s", tableName);
// Execute the drop table statement
if (SQLExecDirect(hstmt, (UCHAR FAR *) sqlStmt, SQL_NTS) != SQL_SUCCESS)
{
  // Check for sqlState = S0002, "Table or view not found".
  // Ignore this error, bomb out on any other error.
  pDb->GetNextError(henv, hdbc, hstmt);
  if (strcmp(pDb->sqlState, "S0002"))
  {
    pDb->DispNextError();                       // Displayed error retrieved
    pDb->DispAllErrors(henv, hdbc, hstmt);      // Display all other errors, if any
    pDb->RollbackTrans();                       // Rollback the transaction
    CloseCursor();                              // Close the cursor
    return(FALSE);                              // Return Failure
  }
}
```

wxDB::ExecSql()

Declaration

Bool ExecSql(char *pSqlStmt)

Arguments

pSqlStmt	Pointer to the SQL statement to be executed.

Comments

This member extends the wxDB class and allows you to build and execute ANY VALID SQL statement against the data source. This allows you to extend the class library by being able to issue any SQL statement that the data source is capable of processing.

wxDB::GetColumns()

Declaration

CcolInf *GetColumns(char *tableName[])

Arguments

tableName	An array of pointers to table names you wish to obtain column information about. The last element of this array must be a NULL string.

Comments

This function returns an array of CcolInf structures. This allows you to obtain information regarding the columns of your table(s). See the declaration of the CcolInf structure for more details regarding the information returned. If no columns were found, or an error occured, this pointer will be zero (null). THE CALLING FUNCTION IS RESPONSIBLE FOR DELETING THE MEMORY RETURNED WHEN IT IS FINISHED WITH IT.

Example

```
char *tableList[] = {“PARTS”, 0};
CcolInf *colInf = pDb->GetColumns(tableList);
if (colInf)
{
   // Use the column inf
   .......
   // Destroy the memory
   delete [] colInf;
 }
```

wxDB::GetDatabaseName()

Declaration

char *GetDatabaseName(void)

Comments

Returns the name of the database engine.

wxDB::GetDataSource()

Declaration

char *GetDataSource(void)

Comments

Returns the ODBC datasource name.

wxDB::GetHDBC()

Declaration

HDBC GetHDBC(void)

Comments

Returns the ODBC handle to the database connection.

wxDB::GetHENV()

Declaration

HENV GetHENV(void)

Comments

Returns the ODBC environment handle.

wxDB::GetHSTMT()

Declaration

HSTMT GetHSTMT(void)

Comments

Returns the ODBC statement handle associated with this database connection.

wxDB::GetNextError()

Declaration

```
Bool GetNextError(HENV aHenv,
                  HDBC aHdbc = SQL_NULL_HDBC,
                  HSTMT aHstmt = SQL_NULL_HSTMT)
```

Arguments

aHenv	A handle to the ODBC environment.
aHdbc	A handle to the ODBC connection. Pass this in if the ODBC function call that erred out required a hdbc or hstmt argument.
AHstmt	A handle to the ODBC statement being executed against. Pass this in if the ODBC function call that erred out required a hstmt argument.

Comments

see wxDB::DispNextError()

Example

see wxDB::DispNextError()

wxDB::GetPassword()

Declaration

char *GetPassword(void)

Comments

Returns the password used to connect to the datasource.

wxDB::GetUsername()

Declaration

char *GetUsername(void)

Comments

Returns the user name connected to the datasource.

wxDB::Grant()

Declaration

Bool Grant(int privileges, char *tableName, char *userList = "PUBLIC");

Arguments

privileges	Use this argument to select which privileges you want to grant. Pass DB_GRANT_ALL to grant all privileges. To grant individual privileges pass one or more of the following OR'd together: DB_GRANT_SELECT, DB_GRANT_INSERT, DB_GRANT_UPDATE and DB_GRANT_DELETE.
tableName	The name of the table you wish to grant privileges on.
userList	A comma delimited list of users to grant the privileges to. If this argument is not passed in, the privileges will be given to the general PUBLIC.

Comments

Use this member function to GRANT privileges to your tables to other database users.

Example

db.Grant(DB_GRANT_SELECT | DB_GRANT_INSERT, “PARTS”, “mary, sue”);

wxDB::IsOpen()

Declaration

Bool IsOpen(void)

Comments

Indicates if the datasource was successfully opened.

wxDB::Open()

Declaration

Bool Open(char *Dsn, char *Uid, char *AuthStr)

Arguments

Dsn	Data source name. The name of the ODBC data source as assigned when the data source is initially set up through the ODBC data source manager.
Uid	User ID. The name (ID) of the user you wish to connect to the data source as. The user name (ID) determines what objects you have access to in the RDBMS and what RDBMS privileges you have. Privileges include being able to create new objects, update objects, delete objects and so on. Users and privileges are normally administered by the database administrator.
AuthStr	The password for the user name (ID).

Comments

After the wxDB object is created, it must then be opened. When you open a data source, you must pass in three pieces of information. The data source name, user name (ID) and the password for the user. No database activity on the data source can be performed until it is opened. This would normally be done at program startup and the data source would remain open for the duration of the program run. Note: It is possible to have multiple data sources open at the same time to support distributed database connections.

Example

```
wxDB sampleDB(Db.Henv);
if (! sampleDB.Open(“Oracle 7.1 HP/UX”, “fastpic”, “maui”))
{
  // Error opening data source
}
```

wxDB::RollbackTrans()

Declaration

Bool RollbackTrans(void)

Comments

Call this member function to “rollback” changes made to the database. Transactions begin implicitly as soon as you make a change to the database. At any time thereafter, you can save your work to the database (“Commit”) or roll back all of your changes (“Rollback”). Calling this member function rolls back **all** open transactions on this ODBC connection.

See wxDB::CommitTrans() for “Special Note re: Cursors”

wxDB::SqlLog()

Declaration

Bool SqlLog(enum sqlLog state, char *filename = "sqllog.txt", Bool append = FALSE);

Arguments

state	Either ***sqlLogOFF*** *or* ***sqlLogON***. Turns logging of SQL commands sent to the data source OFF or ON.
filename	OPTIONAL argument. Name of the file to which the log text is to be written
append	OPTIONAL argument. Whether the file should be appended to or overwritten

Comments

When called with ***sqlLogON***, all commands sent to the data source engine are logged to the file specified by “filename”. Logging is done by embedded WriteSqlLog() calls in the database classes, or may be manually logged by adding calls to WriteSqlLog() in your own source code.

When called with ***sqlLogOFF***, the logging file is closed, and any calls to WriteSqlLog() are ignored.

wxDB::TableExists()

Declaration

Bool TableExists(char *tableName)

Comments

Checks the ODBC data source for the existence of a table. ‘tableName’ may refer to a table, view, alias or synonym.

wxDB::TranslateSqlState()

Declaration

int TranslateSqlState(char *SQLState)

Comments

Converts an ODBC sqlstate to an internal error code.

wxDB::WriteSqlLog()

Declaration

Bool WriteSqlLog(char *logMsg)

Arguments

logMsg	Free form string to be written to the log file

Comments

Very useful debugging tool that may be turned on/off during run time. The passed in string “logMsg” will be written to a log file if SQL logging is turned on (see SqlLog() for details on turning logging on/off) If logging is off when a call to WriteSqlLog() is made, the function returns FALSE without performing the requested log.

Public Member Variables

wxDB::cbErrorMsg

Declaration

SWORD cbErrorMsg

Comments

This member variable is populated as a result of calling wxDB::GetNextError(). Contains the count of bytes in the wxDB::errorMsg string.

wxDB::dbInf

Declaration

```
struct
{
   char       dbmsName[40];              // Name of the dbms product
   char       dbmsVer[20];               // Version # of the dbms product
   char       driverName[40];            // Driver name
   char       odbcVer[20];               // ODBC version
   char       drvMgrOdbcVer[20];// ODBC version of the driver manager
   char       driverVer[40];             // Driver version
   char       serverName[40];            // Server Name, typically a connect string
   char       databaseName[128];         // Database filename
   char       outerJoins[2];             // Indicates if data source supports outer joins
   char       procedureSupport[2];// Indicates if data source supports stored procedures
   UWORD      maxConnections;            // Max # of connections the data source supports
   UWORD      maxStmts;                  // Maximum # of HSTMTs per HDBC
   UWORD      apiConfLvl;                // ODBC API conformance level
   UWORD      cliConfLvl;                // Indicates whether data source is SAG compliant
   UWORD      sqlConfLvl;                // SQL conformance level
   UWORD      cursorCommitBehavior;      // Indicates how cursors are affected by a db commit
   UWORD      cursorRollbackBehavior;    // Indicates how cursors are affected by a db rollback
   UWORD      supportNotNullClause;      // Indicates if data source supports NOT NULL clause
   char       supportIEF[2];             // Integrity Enhancement Facility (Ref. Integrity)
   UDWORD  txnIsolation;          // Default trans. isolation level supported by driver
   UDWORD  txnIsolationOptions;          // Transaction isolation level options available
   UDWORD  fetchDirections;              // Fetch directions supported
   UDWORD  lockTypes;                    // Lock types supported in SQLSetPos
   UDWORD  posOperations;                // Position operations supported in SQLSetPos
   UDWORD  posStmts;                     // Position statements supported
   UDWORD  scrollConcurrency;            // Concurrency control options supported
   UDWORD  scrollOptions;                // Scroll Options supported for scrollable cursors
   UDWORD  staticSensitivity;            // Can additions, deletions and updates be detected?
   UWORD      txnCapable;                // Indicates if the data source supports transactions
   UDWORD  loginTimeout;                 // Number seconds to wait for a login request
} dbInf;
```

Comments

When the data source is opened, all of the information contained in the dbInf structure is queried from the data source. This information is used almost exclusively within the ODBC class library. Where there is a need for this information outside of the class library a member function such as wxTable::IsCursorClosedOnCommit() has been added for ease of use.

wxDB::DB_STATUS

Declaration

int DB_STATUS

Comments

The last ODBC error that occured on this data connection. Possible codes are:

DB_ERR_GENERAL_WARNING	// SqlState = '01000'
DB_ERR_DISCONNECT_ERROR	// SqlState = '01002'
DB_ERR_DATA_TRUNCATED	// SqlState = '01004'
DB_ERR_PRIV_NOT_REVOKED	// SqlState = '01006'
DB_ERR_INVALID_CONN_STR_ATTR	// SqlState = '01S00'
DB_ERR_ERROR_IN_ROW	// SqlState = '01S01'
DB_ERR_OPTION_VALUE_CHANGED	// SqlState = '01S02'
DB_ERR_NO_ROWS_UPD_OR_DEL	// SqlState = '01S03'
DB_ERR_MULTI_ROWS_UPD_OR_DEL	// SqlState = '01S04'
DB_ERR_WRONG_NO_OF_PARAMS	// SqlState = '07001'
DB_ERR_DATA_TYPE_ATTR_VIOL	// SqlState = '07006'
DB_ERR_UNABLE_TO_CONNECT	// SqlState = '08001'
DB_ERR_CONNECTION_IN_USE	// SqlState = '08002'
DB_ERR_CONNECTION_NOT_OPEN	// SqlState = '08003'
DB_ERR_REJECTED_CONNECTION	// SqlState = '08004'
DB_ERR_CONN_FAIL_IN_TRANS	// SqlState = '08007'
DB_ERR_COMM_LINK_FAILURE	// SqlState = '08S01'
DB_ERR_INSERT_VALUE_LIST_MISMATCH	// SqlState = '21S01'
DB_ERR_DERIVED_TABLE_MISMATCH	// SqlState = '21S02'
DB_ERR_STRING_RIGHT_TRUNC	// SqlState = '22001'
DB_ERR_NUMERIC_VALUE_OUT_OF_RNG	// SqlState = '22003'
DB_ERR_ERROR_IN_ASSIGNMENT	// SqlState = '22005'
DB_ERR_DATETIME_FLD_OVERFLOW	// SqlState = '22008'
DB_ERR_DIVIDE_BY_ZERO	// SqlState = '22012'
DB_ERR_STR_DATA_LENGTH_MISMATCH	// SqlState = '22026'
DB_ERR_INTEGRITY_CONSTRAINT_VIOL	// SqlState = '23000'
DB_ERR_INVALID_CURSOR_STATE	// SqlState = '24000'
DB_ERR_INVALID_TRANS_STATE	// SqlState = '25000'
DB_ERR_INVALID_AUTH_SPEC	// SqlState = '28000'
DB_ERR_INVALID_CURSOR_NAME	// SqlState = '34000'
DB_ERR_SYNTAX_ERROR_OR_ACCESS_VIOL	// SqlState = '37000'
DB_ERR_DUPLICATE_CURSOR_NAME	// SqlState = '3C000'
DB_ERR_SERIALIZATION_FAILURE	// SqlState = '40001'
DB_ERR_SYNTAX_ERROR_OR_ACCESS_VIOL2	// SqlState = '42000'
DB_ERR_OPERATION_ABORTED	// SqlState = '70100'
DB_ERR_UNSUPPORTED_FUNCTION	// SqlState = 'IM001'
DB_ERR_NO_DATA_SOURCE	// SqlState = 'IM002'
DB_ERR_DRIVER_LOAD_ERROR	// SqlState = 'IM003'

DB_ERR_SQLALLOCENV_FAILED	// SqlState = 'IM004'
DB_ERR_SQLALLOCCONNECT_FAILED	// SqlState = 'IM005'
DB_ERR_SQLSETCONNECTOPTION_FAILED	// SqlState = 'IM006'
DB_ERR_NO_DATA_SOURCE_DLG_PROHIB	// SqlState = 'IM007'
DB_ERR_DIALOG_FAILED	// SqlState = 'IM008'
DB_ERR_UNABLE_TO_LOAD_TRANSLATION_DLL	// SqlState = 'IM009'
DB_ERR_DATA_SOURCE_NAME_TOO_LONG	// SqlState = 'IM010'
DB_ERR_DRIVER_NAME_TOO_LONG	// SqlState = 'IM011'
DB_ERR_DRIVER_KEYWORD_SYNTAX_ERROR	// SqlState = 'IM012'
DB_ERR_TRACE_FILE_ERROR	// SqlState = 'IM013'
DB_ERR_TABLE_OR_VIEW_ALREADY_EXISTS	// SqlState = 'S0001'
DB_ERR_TABLE_NOT_FOUND	// SqlState = 'S0002'
DB_ERR_INDEX_ALREADY_EXISTS	// SqlState = 'S0011'
DB_ERR_INDEX_NOT_FOUND	// SqlState = 'S0012'
DB_ERR_COLUMN_ALREADY_EXISTS	// SqlState = 'S0021'
DB_ERR_COLUMN_NOT_FOUND	// SqlState = 'S0022'
DB_ERR_NO_DEFAULT_FOR_COLUMN	// SqlState = 'S0023'
DB_ERR_GENERAL_ERROR	// SqlState = 'S1000'
DB_ERR_MEMORY_ALLOCATION_FAILURE	// SqlState = 'S1001'
DB_ERR_INVALID_COLUMN_NUMBER	// SqlState = 'S1002'
DB_ERR_PROGRAM_TYPE_OUT_OF_RANGE	// SqlState = 'S1003'
DB_ERR_SQL_DATA_TYPE_OUT_OF_RANGE	// SqlState = 'S1004'
DB_ERR_OPERATION_CANCELLED	// SqlState = 'S1008'
DB_ERR_INVALID_ARGUMENT_VALUE	// SqlState = 'S1009'
DB_ERR_FUNCTION_SEQUENCE_ERROR	// SqlState = 'S1010'
DB_ERR_OPERATION_INVALID_AT_THIS_TIME	// SqlState = 'S1011'
DB_ERR_INVALID_TRANS_OPERATION_CODE	// SqlState = 'S1012'
DB_ERR_NO_CURSOR_NAME_AVAIL	// SqlState = 'S1015'
DB_ERR_INVALID_STR_OR_BUF_LEN	// SqlState = 'S1090'
DB_ERR_DESCRIPTOR_TYPE_OUT_OF_RANGE	// SqlState = 'S1091'
DB_ERR_OPTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1092'
DB_ERR_INVALID_PARAM_NO	// SqlState = 'S1093'
DB_ERR_INVALID_SCALE_VALUE	// SqlState = 'S1094'
DB_ERR_FUNCTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1095'
DB_ERR_INF_TYPE_OUT_OF_RANGE	// SqlState = 'S1096'
DB_ERR_COLUMN_TYPE_OUT_OF_RANGE	// SqlState = 'S1097'
DB_ERR_SCOPE_TYPE_OUT_OF_RANGE	// SqlState = 'S1098'
DB_ERR_NULLABLE_TYPE_OUT_OF_RANGE	// SqlState = 'S1099'
DB_ERR_UNIQUENESS_OPTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1100'
DB_ERR_ACCURACY_OPTION_TYPE_OUT_OF_RANGE	// SqlState = 'S1101'
DB_ERR_DIRECTION_OPTION_OUT_OF_RANGE	// SqlState = 'S1103'
DB_ERR_INVALID_PRECISION_VALUE	// SqlState = 'S1104'
DB_ERR_INVALID_PARAM_TYPE	// SqlState = 'S1105'
DB_ERR_FETCH_TYPE_OUT_OF_RANGE	// SqlState = 'S1106'
DB_ERR_ROW_VALUE_OUT_OF_RANGE	// SqlState = 'S1107'
DB_ERR_CONCURRENCY_OPTION_OUT_OF_RANGE	// SqlState = 'S1108'
DB_ERR_INVALID_CURSOR_POSITION	// SqlState = 'S1109'
DB_ERR_INVALID_DRIVER_COMPLETION	// SqlState = 'S1110'
DB_ERR_INVALID_BOOKMARK_VALUE	// SqlState = 'S1111'
DB_ERR_DRIVER_NOT_CAPABLE	// SqlState = 'S1C00'
DB_ERR_TIMEOUT_EXPIRED	// SqlState = 'S1T00'

wxDB::errorList

Declaration

char errorList[DB_MAX_ERROR_HISTORY][DB_MAX_ERROR_MSG_LEN]

Comments

The last *n* ODBC errors that have occured on this database connection.

wxDB::errorMsg

Declaration

char   errorMsg[SQL_MAX_MESSAGE_LENGTH]

Comments

This member variable is populated as a result of calling wxDB::GetNextError(). Contains the ODBC error message text.

wxDB::hdbc

Declaration

HDBC hdbc

Comments

Handle to the database connection. Needed only if calling ODBC functions directly.

wxDB::henv

Declaration

HENV henv

Comments

Handle to the ODBC environment. Needed only if calling ODBC functions directly.

wxDB::hstmt

Declaration

HSTMT hstmt

Comments

ODBC statement handle created automatically with each wxDB object. Needed only if calling ODBC functions directly.

wxDB::nativeError

Declaration

SDWORD nativeError

Comments

This member variable is populated as a result of calling wxDB::GetNextError(). This is the native error code returned from the RDBMS engine.

wxDB::silent

Declaration

Bool silent

Comments

Toggles the error reporting mode of the ODBC class library. When this member variable is TRUE, errors are not reported to the user automatically. When this variable is FALSE, every error is displayed to the user in a text message or pop-up dialog window. Default value is TRUE.

wxDB::sqlState

Declaration

char sqlState[20]

Comments

This member variable is populated as a result of calling wxDB::GetNextError(). This is the ODBC error code.

wxDB::typeInfDate

Declaration

SqlTypeInfo typeInfDate

Comments

A structure containing information about the logical "date" data type. Primarily used by the ODBC classes to convert data between ODBC and C native formats.

wxDB::typeInfFloat

Declaration

SqlTypeInfo typeInfFloat

Comments

A structure containing information about the logical “float” data type. Primarily used by the ODBC classes to convert data between ODBC and C native formats.

wxDB::typeInfInteger

Declaration

SqlTypeInfo typeInfInteger

Comments

A structure containing information about the logical “Integer” data type. Primarily used by the ODBC classes to convert data between ODBC and C native formats.

wxDB::typeInfVarchar

Declaration

SqlTypeInfo typeInfVarchar

Comments

A structure containing information about the logical “varchar” data type. Primarily used by the ODBC classes to convert data between ODBC and C native formats. Varchar stands for variable length character strings.

wxTable Class

Constructors/Destructors

wxTable::wxTable()

Declaration

wxTable(wxDB *pwxDB, const char *tblName, const int nCols, const char *queryTblName = 0)

Arguments

pwxDB	Pointer to the wxDB object containing this table.
tblName	The name of the table in the RDBMS.
ncols	The number of columns in the table. Do NOT include the ROWID column.
queryTblName	Optional argument. The name of the table or view to base your queries on. This argument allows you to specify a table/view other than the base table for this object to base your queries on. This allows you to query on a view for example, but all of the INSERT, UPDATE and DELETES will still be performed on the base table for this wxTable object. Basing your queries on a view can provide a substantial performance increase in cases where your queries involve many tables with many joins.

Comments

The derived wxTable constructor is where you describe your table layout to the ODBC class library. This description includes column definitions for each column in your table and specification of which columns are key fields.

The key fields will make up the primary index on the table. The column definitions will be scanned in order looking for columns that have been tagged as keys. These columns will make up the primary index specification when the table is created. If you're primary index is made up of multiple columns and it's critical that they are in a certain order in the index designation, then they should be listed in the appropriate order in the column definition structure. For example, let's say you have a two part index made up of the columns warehouse and part#. It may be critical to performance that the index is part#/warehouse, where part# is the first column in the index and warehouse is the second. Currently, the only way to achieve this is by placing part# before warehouse in the list of column definitions (CcolDef[]).

To create secondary indexes on your table, see wxTable::CreateIndex.

For a complete description of the column specifications, see the section on the CcolDef structure.

Example

```
#define PARTS_TABLE_NAME    "PARTS"
#define PARTS_NO_COLS       5
#define PART_NO_LEN         10
#define PART_DESC_LEN       40

        // Cparts declaration included here for clarity
class Cparts : public wxTable
{
```

```
public:

    struct
    {
        char                  partNo[PART_NO_LEN+1];
        char                  partDesc[PART_DESC_LEN+1];
        float                 qtyOnHand;
        ULONG                 serialNo;
        TIMESTAMP_STRUCT      expDate;
    } PartsRec;

    Cparts(wxDB *pwxDB);

};
```

Note that the WxTable constructor is called here. At this point adequate memory is allocated for the colDefs array based on the number of columns parameter. Do NOT include ROWID in this number of columns!

```
    // Cparts constructor
Cparts::Cparts(wxDB *pwxDB) : wxTable(pwxDB, PARTS_TABLE_NAME, PARTS_NO_COLS)
{
        // Initialize the column definitions for this table.  The wxTable constructor
        // allocated ample memory for the no. of columns specified.
    strcpy(colDefs[0].ColName, "PART_NO");
    colDefs[0].DbDataType   = DB_DATA_TYPE_VARCHAR;
    colDefs[0].PtrDataObj   = PartsRec.partNo;
    colDefs[0].SqlCtype     = SQL_C_CHAR;
    colDefs[0].SzDataObj    = PART_NO_LEN;
    colDefs[0].KeyField     = TRUE;
    colDefs[0].Updateable   = FALSE;
```

For each column in your table, describe it to the ODBC class library.

```
    strcpy(colDefs[1].ColName, "PART_DESC");
    colDefs[1].DbDataType   = DB_DATA_TYPE_VARCHAR;
    colDefs[1].PtrDataObj   = PartsRec.partDesc;
    colDefs[1].SqlCtype     = SQL_C_CHAR;
    colDefs[1].SzDataObj    = PART_DESC_LEN;
    colDefs[1].KeyField     = FALSE;
    colDefs[1].Updateable   = TRUE;

    strcpy(colDefs[2].ColName, "QTY_ON_HAND");
    colDefs[2].DbDataType   = DB_DATA_TYPE_FLOAT;
    colDefs[2].PtrDataObj   = &PartsRec.qtyOnHand;
    colDefs[2].SqlCtype     = SQL_C_FLOAT;
    colDefs[2].SzDataObj    = sizeof(PartsRec.qtyOnHand);
    colDefs[2].KeyField     = FALSE;
    colDefs[2].Updateable   = TRUE;

    strcpy(colDefs[3].ColName, "SERIAL_NO");
    colDefs[3].DbDataType   = DB_DATA_TYPE_INTEGER;
    colDefs[3].PtrDataObj   = &PartsRec.serialNo;
    colDefs[3].SqlCtype     = SQL_C_ULONG;
    colDefs[3].SzDataObj    = sizeof(PartsRec.serialNo);
    colDefs[3].KeyField     = FALSE;
    colDefs[3].Updateable   = TRUE;

    strcpy(colDefs[4].ColName, "EXPIRATION_DATE");
    colDefs[4].DbDataType   = DB_DATA_TYPE_DATE;
```

```
        colDefs[4].PtrDataObj       = &PartsRec.expDate;
        colDefs[4].SqlCtype         = SQL_C_TIMESTAMP;
        colDefs[4].SzDataObj        = sizeof(PartsRec.expDate);
        colDefs[4].KeyField         = FALSE;
        colDefs[4].Updateable       = TRUE;
}
```

wxTable::~wxTable()

Declaration

~wxTable()

Comments

wxTable destructor. Frees up memory allocated for column definitions and frees all ODBC statement handles associated with the wxTable object. For wxTable objects dynamically created with “new”, be sure to “delete” the object so that the destructor is called and memory is freed up.

Public Member Functions

wxTable::CanSelectForUpdate()

Declaration

Bool CanSelectForUpdate(void)

Comments

Use this function to determine if the data source supports SELECT … FOR UPDATE. When the keywords "FOR UPDATE" are included as part of your SQL SELECT statement, all records retrieved from the query are locked. Note however that not all data sources support the "FOR UPDATE" clause, so you must use this member function to determine if they do or not.

FASTPIC will be using a logical lock manager since locks managed through ODBC are very limited. Consequently, at this time, I see no need to perform database locks using the "FOR UPDATE" clause. It cannot hurt, and may be somewhat safer though to add the "FOR UPDATE" clause if you know you're going to update or delete every record in your query. This will give you the added protection of an RDBMS lock as well as a logical lock applied by your application. The "FOR UPDATE" clause is handled automatically unless you QueryBySqlStmt(). If you are querying by manually writing a SQL SELECT statement (QueryBySqlStmt()), is the only time you may need to know whether the data source can CanSelectForUpdate().

wxTable::CanUpdByROWID()

Declaration

Bool CanUpdByROWID(void)

Comments

Every Oracle table has a hidden column named ROWID. This is a pointer to the physical location of the record in the data store and allows for very fast updates and deletes. The key is to retrieve this ROWID during your query so it's available during an update or delete operation.

This is always handled by the class library except in the case of QueryBySqlStatement(). Since you are passing in the SQL SELECT statement, it's up to you to include the ROWID column in your query. If you don't, the application will still work, but may not be as optimized. The ROWID is always the last column in the column list in your SQL SELECT statement. The ROWID is not a column in the normal sense and should not be considered part of the column definitions for the wxTable object.

Note that the decision to include the ROWID in your SQL SELECT statement must be deferred until runtime since it depends on whether you are connected to an Oracle data source or not.

Example

```
        // Incomplete code sample
wxTable parts;

if (parts.CanUpdByROWID())
        // Note that the ROWID must always be the last column selected
```

```
   sqlStmt = “SELECT PART_NO, PART_DESC, ROWID” FROM PARTS”;
else
   sqlStmt = “SELECT PART_NO, PART_DESC” FROM PARTS”;
```

wxTable::ClearMemberVars()

Declaration

void ClearMemberVars(void)

Comments

Sets all of the “columns” for the wxTable object equal to zero. In the case of a string, zero is copied to the first byte of the string. This is useful before calling QueryMatching() or DeleteMatching() since these functions build their WHERE clauses from non-zero columns. To call QueryMatching() or DeleteMatching() you may:

1) ClearMemberVars()
2) Assign columns values you wish to match on
3) Call QueryMatching() or DeleteMatching()

wxTable::CloseCursor()

Declaration

Bool CloseCursor(HSTMT cursor)

Arguments

cursor	The cursor to be closed.

Comments

Closes the selected cursor associated with the wxTable object. Used almost exclusively within the ODBC class library.

wxTable::Count()

Declaration

ULONG Count(void)

Comments

Returns the number of records in the current record set. Based on the wxTable::where clause. This function can be executed before or after the query to obtain the count of records.

wxTable::CreateIndex()

Declaration

Bool CreateIndex(char * idxName, Bool unique, int noIdxCols, CidxDef *pIdxDefs)

Arguments

idxName	Name of the Index. Must be unique.
unique	Indicates if the index is unique.
noIdxCols	The number of columns in the index.
pIdxDefs	A pointer to an array CidxDef structures.

Comments

This member function allows you to create secondary (non primary) indexes on your tables. You first create your table, normally specifying a primary index, and then create any secondary indexes on the table. Indexes in relational model are not required. You do not need indexes to look up records in a table or to join two tables together. In the relational model, indexes, if available, provide a quicker means to look up data in a table. To enjoy the performance benefits of indexes, the indexes must be defined on the appropriate columns and your SQL code must be written in such a way as to take advantage of those indexes. To create a secondary index(es) on your wxTable object, simply call wxTable::CreateIndex.

The first parameter, index name, must be unique and should be given a meaningful name. I would suggest including the table name as a prefix in the index name (e.g. PARTS_IDX1). This will allow you to easily view all of the indexes defined for a given table.

The second parameter indicates if the index is unique or not. Uniqueness is enforced at the RDBMS level so that duplicates cannot be inserted into the table for unique indices.

In the third parameter, specify how many columns are in your index.

The fourth parameter is where you specify what columns make up the index using the CidxDef structure. For each column in the index, you must specify two things, the column name and the sort order (ascending / descending). See the example to see how you would build and pass in the CidxDef structure.

Example

```
CidxDef  idxDef[2];
strcpy(idxDef[0].ColName, "PART_DESC");
idxDef[0].Ascending = TRUE;
strcpy(idxDef[1].ColName, "SERIAL_NO");
idxDef[1].Ascending = FALSE;
parts->CreateIndex("PARTS_IDX1", TRUE, 2, idxDef);
```

wxTable::CreateTable()

Declaration

Bool CreateTable(void)

Comments

This function creates the table and primary index (if any) in the database associated with the connected data source. The owner of these objects will be the user id that what given to the wxDB::Open() member function. The objects will be created in the default schema for that user.

In your derived wxTable object constructor, the columns and primary index of the table are described through the CcolDef structure. wxTable::CreateTable uses this information to create the table and add the primary index. See wxTable::wxTable and CcolDef description for additional information on describing the columns of the table.

Example

```
parts.CreateTable();
```

wxTable::DB_STATUS()

Declaration

```
int DB_STATUS(void) { return(pDb->DB_STATUS); }
```

Comments

Shortcut for grabbing the DB_STATUS member of the wxDB object.

wxTable::Delete()

Declaration

```
Bool Delete(void)
```

Comments

Deletes the current record from the data source. Use wxTable::GetFirst, wxTable::GetLast, wxTable::GetNext and wxTable::GetPrev to position the wxTable object on a valid record. Once positioned on a record, call wxTable::Delete to delete it.

You should lock the record before deleting it to make sure it's not already in use.

Example

```
        // Incomplete code sample to delete part #32
strcpy(parts.PartsRec.partNo, “32”);
parts.QueryOnKeyFields();
parts.GetFirst();
parts.Delete();
```

wxTable::DeleteMatching()

Declaration

```
Bool DeleteMatching(void)
```

Comments

This member function allows you to delete records from your wxTable object by specifying which columns to match on. For example, to delete all parts from container 'xyz', you would do the following:

1) Set all "columns" to zero using wxTable::ClearMemberVars.
2) Set the container column equal to 'xyz'.
3) Call wxTable::DeleteMatching.

The WHERE clause is built by the ODBC class library based on all non-zero columns. This allows you to delete records by matching on any column(s) in your wxTable object, without having to write the SQL WHERE clause.

You should lock the record(s) before deleting them to make sure they're not already in use. This can be achieved by calling wxTable::QueryMatching, and then scroll through the records locking each as you go. After they have all been successfully locked, call wxTable::DeleteMatching.

Example

```
            // Incomplete code sample to delete all parts from container 3
parts.ClearMemberVars();
parts.PartsRec.container = 3;
parts.DeleteMatching();
```

wxTable::DeleteWhere()

Declaration

Bool DeleteWhere(char *pWhereClause)

Arguments

pWhereClause	SQL WHERE clause. This WHERE clause determines which records will be deleted from the wxTable object. Must be compliant with the SQL 92 grammar. Do not include the keyword 'WHERE'.

Comments

This is the most powerful form of the wxTable delete functions. This gives you access to the full power of SQL. You can delete records by submitting a valid SQL WHERE clause to this function. This allows you to perform sophisticated deletions based on multiple criteria and using the full functionality of the SQL language. Note however, that you are limited to deleting records from the table associated with this wxTable object only.

Example

```
            // Delete parts 1 thru 10 from containers 'X', 'Y' and 'Z' that are magenta in color
parts.DeleteWhere("(PART_NO BETWEEN 1 AND 10) AND
                   CONTAINER IN ('X', 'Y', 'Z') AND
                   UPPER(COLOR) = 'MAGENTA'");
```

wxTable::GetCursor()

Declaration

int GetCursor(void)

Comments

Returns the value of the current cursor for this wxTable object. Returns the value 1 through 5. Cursor 0 is considered a temporary cursor and therefore is not returned as a value from this function. This member allows you to switch to cursor 0 temporarily, and then switch back to the previous cursor without having to know which cursor you were on.

wxTable::GetDeleteStmt()

Declaration

void GetDeleteStmt(char *pSqlStmt, int typeOfDel, char *pWhereClause = 0)

Arguments

pSqlStmt	Pointer to storage for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate DB_MAX_STATEMENT_LEN bytes.
typeOfDel	The type of delete statement being performed. Can be one of three values: DB_DEL_KEYFIELDS, DB_DEL_WHERE or DB_DEL_MATCHING.
pWhereClause	If the typeOfDel = DB_DEL_WHERE, then you must also pass in a SQL WHERE clause in this argument.

Comments

This member function allows you to see what the SQL DELETE statement looks like that the ODBC class library builds. This can be used for debugging purposes if you're having problems executing your SQL statement.

wxTable::GetFirst()

Declaration

Bool GetFirst(void)

Comments

Retrieves the FIRST record in the record set as defined by your query. Before you can retrieve records, you must query them using wxTable::Query, wxTable::QueryOnKeyFields, wxTable::QueryMatching or wxTable::QueryBySqlStmt.

wxTable::GetLast()

Declaration

Bool GetLast(void)

Comments

Retrieves the LAST record in the record set as defined by your query. Before you can retrieve records, you must query them using wxTable::Query, wxTable::QueryOnKeyFields, wxTable::QueryMatching or wxTable::QueryBySqlStmt.

wxTable::GetNext()

Declaration

Bool GetNext(void)

Comments

Retrieves the NEXT record in the record set as defined by your query. Before you can retrieve records, you must query them using wxTable::Query, wxTable::QueryOnKeyFields, wxTable::QueryMatching or wxTable::QueryBySqlStmt.

wxTable::GetNext returns FALSE on EOF.

wxTable::GetPrev()

Declaration

Bool GetPrev(void)

Comments

Retrieves the PREVIOUS record in the record set as defined by your query. Before you can retrieve records, you must query them using wxTable::Query, wxTable::QueryOnKeyFields, wxTable::QueryMatching or wxTable::QueryBySqlStmt.

wxTable::GetPrev returns FALSE on BOF.

wxTable::GetRowNum()

Declaration

UWORD GetRowNum(void)

Comments

Returns the ODBC row number for performing positioned updates and deletes. Currently, this function is not being used within the ODBC class library and may be a candidate for removal if no use is found for it.

wxTable::GetSelectStmt()

Declaration

void GetSelectStmt(char *pSqlStmt, int typeOfSelect , Bool distinct)

Arguments

pSqlStmt	Pointer to storage for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate DB_MAX_STATEMENT_LEN bytes.
typeOfSelect	The type of select statement being performed. Can be one of four values: DB_SELECT_KEYFIELDS, DB_SELECT_WHERE, DB_SELECT_MATCHING or DB_SELECT_STATEMENT.
distinct	Select distinct records only?

Comments

This member function allows you to see what the SQL SELECT statement looks like that the ODBC class library builds. This can be used for debugging purposes if you're having problems executing your SQL statement.

wxTable::GetUpdateStmt()

Declaration

void GetUpdateStmt(char *pSqlStmt, int typeOfUpd, char *pWhereClause = 0)

Arguments

pSqlStmt	Pointer to storage for the SQL statement retrieved. To be sure you have adequate space allocated for the SQL statement, allocate DB_MAX_STATEMENT_LEN bytes.
typeOfUpd	The type of update statement being performed. Can be one of two values: DB_UPD_KEYFIELDS or DB_UPD_WHERE.
pWhereClause	If the typeOfUpd = DB_UPD_WHERE, then you must also pass in a SQL WHERE clause in this argument.

Comments

This member function allows you to see what the SQL UPDATE statement looks like that the ODBC class library builds. This can be used for debugging purposes if you're having problems executing your SQL statement.

wxTable::GetWhereClause()

Declaration

void GetWhereClause(char *pWhereClause, int typeOfWhere , char *qualTableName=0)

Arguments

pWhereClause	Pointer to storage for the SQL WHERE clause retrieved.
typeOfWhere	The type of where clause to generate. One of two values. Either DB_WHERE_KEYFIELDS or DB_WHERE_MATCHING.

qualTableName	OPTIONAL argument. Prepended to all base table column names. For use when a wxTable::from clause has been specified, to clarify which table a column name reference belongs to.

Comments

This member function allows you to see what the SQL WHERE clause looks like that the ODBC class library builds. This can be used for debugging purposes if you are having problems executing your SQL statements.

wxTable::Insert()

Declaration

int Insert(void)

Comments

Inserts a new record into the wxTable object. Uses the values in the member variables ("columns") of the wxTable object for the values to insert into the new record.

Insert() can return one of three values:

DB_SUCCESS	Record inserted successfully (value = 1)
DB_FAILURE	Insert failed (value = 0)
DB_ERR_INTEGRITY_CONSTRAINT_VIOL	The insert failed due to an integrity constraint violation (value = 23000). This occurs when a duplicate non-unique index entry is attempted.

Example

```
strcpy(parts.PartRec.PartNo, "10");
strcpy(parts.PartRec.PartDesc, "Part #10");
parts.PartRec.QtyOnHand = 10.50;
parts.Insert();
```

wxTable::IsColNull()

Declaration

Bool IsColNull(int colNo)

Arguments

colNo	The column number of the column as defined by the wxTable constructor.

Comments

Used primarily in the ODBC class library to determine if a column is equal to "NULL". Works for all data types supported by the ODBC class library.

wxTable::IsCursorClosedOnCommit()

Declaration

Bool IsCursorClosedOnCommit(void)

Comments

Returns true if the cursor associated with this wxTable object is closed after a commit or rollback operation. Returns false otherwise.

See wxDB::CommitTrans() for “Special Note re: Cursors”

wxTable::Open()

Declaration

Bool Open(void)

Comments

Every wxTable object must be opened before it can be used. Call the wxTable::Open member function to open the table object.

Example

```
Cparts *parts = new Cparts(&sampleDB);
if (! parts->Open())
{
        // Error opening parts table
}
```

wxTable::operator++

Declaration

Bool operator++(int)

Comments

Synonym for wxTable::GetNext.

Example

```
        // Query the table
parts->Query();

        // Scroll forward through the records displaying each record as you go
while((*parts)++)
  dispPart(parts);
```

```
            // Scroll backward through the records displaying each record as you go
while((*parts)--)
  dispPart(parts);
```

wxTable::operator--

Declaration

Bool operator--(int)

Comments

Synonym for wxTable::GetPrev.

Example

```
            // Query the table
parts->Query();

            // Scroll forward through the records displaying each record as you go
while((*parts)++)
  dispPart(parts);

            // Scroll backward through the records displaying each record as you go
while((*parts)--)
  dispPart(parts);
```

wxTable::Query()

Declaration

virtual Bool Query(Bool forUpdate = FALSE, Bool distinct = FALSE)

Arguments

forUpdate	Gives you the option of locking records as they are queried (SELECT … FOR UPDATE). If the RDBMS is not capable of the FOR UPDATE clause, this argument is ignored. *See wxTable:: CanSelectForUpdate for additional information regarding this argument.*
distinct	Allows you to select only distinct values from the query (SELECT DISTINCT … FROM …). The notion of distinct applies to the entire record, not individual columns.

Comments

The wxTable::Query function queries records from the data source based on three other wxTable members: “where” , “orderBy”, and “from”. Set the wxTable::where to filter out records to be retrieved (e.g. All parts in storage device 12). Set the wxTable::orderBy to change the order in which records are retrieved from the database (e.g. Ordered by part number). Set the wxTable::from to allow outer joining of the base table with other tables which share a related field. After each of these is set, call wxTable::Query to fetch the records from the database. This scheme has an advantage if you have to requery your record set frequently in that you only have to set your WHERE, ORDER BY, and FROM clauses once. To refresh the record set, simply issue wxTable::Query as frequently as needed. Please note that this may tax the database server and make your application sluggish if done too frequently or unnecessarily. The base table name is prepended to the column names in the event that the wxTable::from clause is non-null.

The cursor is positioned before the first record in the record set after you issue the query. To retrieve the first record, you must call wxTable::GetFirst or wxTable::GetNext.

This function is defined as “virtual” so that it may be overridden for your own purposes.

Be sure and set the wxTable::where, wxTable::orderBy, and wxTable::from member variables to zero if they are not being used in the query. Otherwise, you may get unexpected results.

Example

```
// Incomplete code sample
parts.where    = “STORAGE_DEVICE = ‘SD98’”;
parts.orderBy = “EXPIRATION_DATE”;
parts.from = 0;
// Query the records based on the where, orderBy and from clauses specified
parts.Query();
// Display all records queried
while(parts++)
  dispPart(&parts);
```

wxTable::QueryBySqlStmt()

Declaration

Bool QueryBySqlStmt(char *pSqlStmt)

Arguments

pSqlStmt	A pointer to the SQL SELECT statement to be executed.

Comments

This is the most powerful form of the query functions available. This member function allows you to write your own custom SQL SELECT statement for retrieving data from the data source. This gives you access to the full power of SQL for performing operations such as scalar functions, aggregate functions, table joins, sub-queries and so on. The requirements of your SELECT statement are the following:

1. Must return the correct number of columns. In your derived wxTable constructor you specify how many columns are in your wxTable object. Your select statement must return exactly that many columns.
2. The columns returned must be of the proper data type. For example, if column 3 is defined to be a float, your SELECT statement must return a float for column 3 (e.g. PRICE * 1.10 to increase the price by 10%).
3. You may include the ROWID in your SELECT statement as the last column selected, if the RDBMS supports it. Use wxTable::CanUpdByROWID to determine if the ROWID can be selected from the data source. If it can, you will get much better performance on updates and deletes by including the ROWID in your SELECT.

Even though data can be selected from multiple tables in your select statement (joins), only your “base” table, defined for your wxTable object, is automatically updated through the ODBC class library. You can select data from multiple tables for display purposes however. Include columns in your wxTable object and mark them as non-updateable (See CcolDef for details). This way columns can be selected and displayed from other tables, but only the base table will be updated automatically through the wxTable::Update function. To update tables other than the base table, use the wxTable::Update(sqlStmt) function.

The cursor is positioned before the first record in the record set after you issue the query. To retrieve the first record, you must call wxTable::GetFirst or wxTable::GetNext.

Example

```
        // Incomplete code sample
strcpy(sqlStmt, “SELECT * FROM PARTS WHERE STORAGE_DEVICE = ‘SD98’
        AND CONTAINER = 12”);
        // Query the records using the SQL SELECT statement above
parts.QueryBySqlStmt(sqlStmt);
        // Display all records queried
while(parts++)
   dispPart(&parts);
```

Example SQL statements

```
        // Table Join returning 3 columns
SELECT part_no, part_desc, sd_name
  from parts, storage_devices
  where parts.storage_device_id = storage_devices.storage_device_id

        // Aggregate function returning total number of parts in container 99
SELECT count(*) from PARTS
  where container = 99

        // Order by clause; ROWID, scalar function
SELECT part_no, substring(part_desc, 1, 10), qty_on_hand + 1, ROWID
  from parts
  where warehouse = 10
  order by part_no desc                   // descending order

        // Subquery
SELECT * from parts
  where container in (select container
                        from storage_devices
                        where device_id = 12)
```

wxTable::QueryMatching()

Declaration

Bool QueryMatching(Bool forUpdate = FALSE, Bool distinct = FALSE)

Arguments

forUpdate	Gives you the option of locking records as they are queried (SELECT … FOR UPDATE). If the RDBMS is not capable of the FOR UPDATE clause, this argument is ignored. *See wxTable:: CanSelectForUpdate for additional information regarding this argument.*
distinct	Allows you to select only distinct values from the query (SELECT DISTINCT … FROM …). The notion of distinct applies to the entire record, not individual columns.

Comments

QueryMatching allows you to query records in your wxTable object by matching "columns" to values. For example: To query part #32, you would set the part# column in your wxTable object to "32" and then call wxTable::QueryMatching. The SQL WHERE clause is built by the ODBC class library based on all non-zero columns in your wxTable object. You can match on one, many or all of your wxTable columns. The base table name is prepended to the column names in the event that the wxTable::from clause is non-null.

Note: the primary difference between this function and QueryOnKeyFields is that this function can query on any column(s) in the wxTable object. Please note however that this may not always be very efficient. Searching on non-indexed columns will always require a full table scan.

The cursor is positioned before the first record in the record set after you issue the query. To retrieve the first record, you must call wxTable::GetFirst or wxTable::GetNext.

Example

```
        // Incomplete code sample
parts.ClearMemberVars();                    // Set all columns to zero
parts.PartRec.storage_device = 10;          // Set columns to query on
parts.PartRec.container = 2;
parts.QueryMatching();                      // Query
        // Display all records queried
while(parts++)
  dispPart(&parts);
```

wxTable::QueryOnKeyFields()

Declaration

Bool QueryOnKeyFields(Bool forUpdate = FALSE, Bool distinct = FALSE)

Arguments

forUpdate	Gives you the option of locking records as they are queried (SELECT … FOR UPDATE). If the RDBMS is not capable of the FOR UPDATE clause, this argument is ignored. *See wxTable:: CanSelectForUpdate for additional information regarding this argument.*
distinct	Allows you to select only distinct values from the query (SELECT DISTINCT … FROM …). The notion of distinct applies to the entire record, not individual columns.

Comments

QueryOnKeyFields provides an easy mechanism to query records in your wxTable object by the primary index column(s). Simply assign the primary index column(s) values and then call this member function to retrieve the record. Note that since primary indexes are always unique, this function implicitly always returns a single record from the database. The base table name is prepended to the column names in the event that the wxTable::from clause is non-null.

The cursor is positioned before the first record in the record set after you issue the query. To retrieve the first record, you must call wxTable::GetFirst or wxTable::GetNext.

Example

```
          // Incomplete code sample
strcpy(parts.PartRec.partNo, “32”);
parts.QueryOnKeyFields();
          // Display all records queried
while(parts++)
  dispPart(&parts);
```

wxTable::Refresh()

Declaration

```
Bool Refresh(void)
```

Comments

Refreshes the current record for the active cursor of the wxTable object. This routine is only guaranteed to work if the table has a **unique primary** index defined for it. Otherwise, more than one record may be fetched and there is no guarantee that the correct record will be refreshed. The table’s columns are refreshed to reflect the current data in the database.

wxTable::SetColDefs()

Declaration

```
void    SetColDefs (int index, char *fieldName, int dataType, void *pData, int cType,
                    int size, Bool keyField = FALSE, Bool upd = TRUE,
                    Bool insAllow = TRUE, Bool derivedCol = FALSE);
```

Arguments

int index	Column number (0 to n-1)
char *fieldName	Column name
int dataType	Logical data type
void *pData	Pointer to the data object
int cType	SQL C Type
nt size	Size in bytes
Bool keyField	Indicates if this column is part of the primary index
Bool upd	Updates allowed on this column?
Bool insAllow	Inserts allowed on this column?
Bool derivedCol	Is this a derived column (non base table-query only)?

For a detailed explanation of these arguments, see the section on CcolDef.

Comments

This member function provides a shortcut for defining the columns in your wxTable object. You would use this in your derived wxTable constructor when describing each of the columns in your wxTable object.

Example

```
            // Long way
strcpy(colDefs[0].ColName, "PART_NO");
colDefs[0].DbDataType    = DB_DATA_TYPE_VARCHAR;
colDefs[0].PtrDataObj    = PartsRec.partNo;
colDefs[0].SqlCtype      = SQL_C_CHAR;
colDefs[0].SzDataObj     = PART_NO_LEN;
colDefs[0].KeyField      = TRUE;
colDefs[0].Updateable    = FALSE;
colDefs[0].InsertAllowed = TRUE;
colDefs[0].DerivedCol    = FALSE;

            // Shortcut
SetColDefs(0, "PART_NO", DB_DATA_TYPE_VARCHAR, PartsRec.partNo,
            SQL_C_CHAR, PART_NO_LEN, TRUE, FALSE,TRUE,FALSE);
```

wxTable::SetCursor()

Declaration

Bool SetCursor(int cursorNo = DB_CURSOR0)

Arguments

cursorNo	Identifies which cursor you'd like to make active. Can be one of the following values: DB_CURSOR0, DB_CURSOR1 or DB_CURSOR2. If no argument is passed, then the scratch cursor (DB_CURSOR0) is used.

Comments

Each wxTable object supports up to 2 active cursors, plus a third "temp" cursor. This allows you to have up to 3 record sets open at the same time against a single wxTable object. This member function allows you to switch between cursors at any time necessary. The member variables of the wxTable object will automatically get refreshed from the current record of the selected cursor, as a result of calling SetCursor(). The primary difference between cursor0 and cursors 1 - 2 is that when you switch to cursor0 it is not recorded in the wxTable::currCursorNo member variable. This way you can switch to a temp. cursor (cursor0), and then switch back to the previous cursor using wxTable::GetCursor().

wxTable::SetQueryTimeout()

Declaration

Bool SetQueryTimeout(UDWORD nSeconds)

Arguments

nSeconds	The number of seconds to wait for the query to complete before timing out.

Comments

Allows you to set the timeout period for queries. Neither Oracle or Access support this function as of yet. Other databases should be evaluated for support before depending on this function working correctly.

wxTable::Update()

Declaration

Bool Update(void)

Comments

Updates the current record in the database with the values in the wxTable “columns”. It identifies the record to update by the primary index column(s). This implies that you cannot modify the primary index column(s) using this function. Instead, you end up updating the wrong record by mistake. Use one of the other two forms of the update function to update the primary index column(s) or delete and reinsert the record using the new primary index values.

To change which columns are updated and which are not, change the Updateable attribute on the column definition (see CcolDef section for details).

Example

```
		// Incomplete code sample
strcpy(parts.PartRec.partNo, “32”);
parts.QueryOnKeyFields();
parts.GetNext();
		// Update part #32
parts.PartRec.qtyOnHand *= 1.10;
parts.Update();
```

wxTable::Update()

Declaration

Bool Update(char *pSqlStmt)

Arguments

pSqlStmt	A pointer to SQL UPDATE statement to be executed.

Comments

This member function allows you full access to SQL for updating records in your database. Write any valid SQL UPDATE statement and submit to this function for execution. This allows you to perform sophisticated updates using the full power of the SQL dialect. Note that you are not tied to only updating your wxTable object through this function. You can update any table in the database for which you have update privileges on.

Example

```
          // Incomplete code sample
strcpy(sqlStmt, “UPDATE parts set qty = 0 where storage_device = 10”);
parts.Update(sqlStmt);
strcpy(sqlStmt, “UPDATE containers set qty = 0 where storage_device = 10”);
parts.Update(sqlStmt);
strcpy(sqlStmt, “UPDATE storage_devices set qty = 0 where storage_device = 10”);
parts.Update(sqlStmt);
```

wxTable::UpdateWhere()

Declaration

Bool UpdateWhere(char *pWhereClause)

Arguments

pWhereClause	A pointer to a valid SQL WHERE clause. Do not include the keyword ‘WHERE’.

Comments

This form of the update function allows updates to the base table of the wxTable object. You determine which records of the wxTable object are updated by passing in a valid SQL WHERE clause. For each record that satisfies the WHERE clause, the record will be updated with values from the wxTable “columns”.

To change which columns are updated and which are not, change the Updateable attribute on the column definition (see CcolDef section for details).

Example

```
          // Incomplete code sample
strcpy(parts.PartRec.partNo, “32”);
parts.QueryOnKeyFields();
parts.GetNext();
          // Update part #32
parts.PartRec.qtyOnHand *= 1.10;
parts.UpdateWhere(“part_no = ‘32’”);
```

Public Member Variables

wxTable::c0

Declaration

HSTMT c0

Comments

Handle to cursor 0, a statement handle for this wxTable object. Needed only if calling ODBC functions directly. This cursor is considered the “temporary” cursor. When you call SetCursor() with no arguments, the cursor is set to c0.

wxTable::c1

Declaration

HSTMT c1

Comments

Handle to cursor 1, a statement handle for this wxTable object. Needed only if calling ODBC functions directly.

wxTable::c2

Declaration

HSTMT c2

Comments

Handle to cursor 2, a statement handle for this wxTable object. Needed only if calling ODBC functions directly.

wxTable::colDefs

Declaration

CcolDef *colDefs

Comments

colDefs is an array of CcolDef structures. ColDefs[n] references the nth column definition for the wxTable object. The memory for the CcolDef structures is allocated in the wxTable constructor. In your derived wxTable object's constructor, you initialize the column definitions for your wxTable object. These column definitions describe to the ODBC class library the columns and primary index of your table object.

See the section "CcolDef" for a detailed description of this structure.

wxTable::from

Declaration

char *from

Comments

SQL FROM clause. Used by the wxTable::Query and wxTable::Count member functions to allow outer joining of records from multiple tables. Do not include the keyword FROM in your "from" clause. If multiple tables are to be specified in the FROM clause, separate the table names with commas. The base table associated with the wxTable instance must not be specified in the FROM clause as it is intrinsically associated.

IF IDENTICAL COLUMN NAMES EXIST IN TWO OR MORE TABLES IN THE FROM CLAUSE (INCLUDING THE BASE TABLE), THEN IF THOSE COLUMN NAMES ARE REFERENCED IN THE "where" OR "orderBy" CLAUSES, THEN THE COLUMN NAME MUST BE PREFACED WITH THE TABLE NAME (i.e. EMPLOYEE.LAST_NAME)

BE CAREFUL NOT TO COPY TO THE ADDRESS OF THIS VARIABLE, SINCE THERE IS NO MEMORY ALLOCATED FOR IT!!!

wxTable::hdbc

Declaration

HDBC hdbc

Comments

Handle to the database connection. Needed only if calling ODBC functions directly.

wxTable::henv

Declaration

HENV henv

Comments

Handle to the ODBC environment. Needed only if calling ODBC functions directly.

wxTable::hstmt

Declaration

HSTMT hstmt

Comments

Handle to the default statement handle for this wxTable object. Needed only if calling ODBC functions directly.

wxTable::hstmtDelete

Declaration

HSTMT hstmtDelete

Comments

Handle to the delete statement handle. Needed only if calling ODBC functions directly.

wxTable::hstmtInsert

Declaration

HSTMT hstmtInsert

Comments

Handle to the insert statement handle. Needed only if calling ODBC functions directly.

wxTable::hstmtUpdate

Declaration

HSTMT hstmtUpdate

Comments

Handle to the update statement handle. Needed only if calling ODBC functions directly.

wxTable::noCols

Declaration

int noCols

Comments

The number of columns in the wxTable object. Set by the constructor.

wxTable::orderBy

Declaration

char *orderBy

Comments

SQL ORDER BY clause. Used by the wxTable::Query member function to order records. Do not include the keywords ORDER BY in your order by clause.

BE CAREFUL NOT TO COPY TO THE ADDRESS OF THIS VARIABLE, SINCE THERE IS NO MEMORY ALLOCATED FOR IT!!!

wxTable::pDb

Declaration

wxDB *pDb

Comments

pDb is a pointer to the wxDB object this wxTable is connected to. This member variable can be used to access wxDB members such as wxDB::CommitTrans() from with your derived wxTable objects.

wxTable::selectForUpdate

Declaration

Bool selectForUpdate

Comments

Currently used internally only.

wxTable::tableName

Declaration

char tableName[DB_MAX_TABLE_NAME_LEN]

Comments

The name of the table in the data source. Set by the wxTable constructor.

wxTable::where

Declaration

char *where

Comments

SQL WHERE clause. Used by the wxTable::Query member function to filter records out during a query. Set this member to a valid SQL WHERE clause, excluding the keyword WHERE.

BE CAREFUL NOT TO COPY TO THE ADDRESS OF THIS VARIABLE, SINCE THERE IS NO MEMORY ALLOCATED FOR IT!!!

CcolDef

Overview

This class (structure) is used to hold information about the columns of your wxTable object. Each instance of this class describes one column in your wxTable object. When you call the wxTable constructor you pass in the number of columns in your wxTable object as an argument. The constructor uses this information to allocate adequate memory for all of the column descriptions in your wxTable object. wxTable::colDefs is a pointer to this chunk of memory. To access the nth column definition of your wxTable object, just reference colDefs[n - 1].

Class Declaration

```
class CcolDef
{
public:
   char          ColName[DB_MAX_COLUMN_NAME_LEN];
   int           DbDataType;
   int           SqlCtype;
   void          *PtrDataObj;
   int           SzDataObj;
   Bool          KeyField;
   Bool          Updateable;
   Bool          InsertAllowed;
   Bool          DerivedCol;
   SDWORD        CbValue;
};          // CcolDef
```

Class Details

ColName Column name

DbDataType The logical data type of the column. Currently four data types are supported:

1. DB_DATA_TYPE_VARCHAR[1]
2. DB_DATA_TYPE_INTEGER
3. DB_DATA_TYPE_FLOAT
4. DB_DATA_TYPE_DATE

[1] Variable length character string

SqlCtype The C data type of the member variable. Should be one of the following values in the SqlCtype column. The C type column shows the equivalent C declaration.

SqlCtype	C type
SQL_C_CHAR	unsigned char FAR *
SQL_C_SSHORT	short int
SQL_C_USHORT	unsigned short int
SQL_C_SLONG	long int
SQL_C_ULONG	unsigned long int
SQL_C_FLOAT	float
SQL_C_DOUBLE	double
SQL_C_DATE	struct tagDATE_STRUCT { [1]

```
            SWORD year;
            UWORD month;
            UWORD day;
}
```

SQL_C_TIME

```
struct tagTIME_STRUCT { [2]
            UWORD hour;
            UWORD minute;
            UWORD second;
}
```

SQL_C_TIMESTAMP

```
struct tagTIMESTAMP_STRUCT { [3]
            SWORD year;
            UWORD month;
            UWORD day;
            UWORD hour;
            UWORD minute;
            UWORD second;
            UDWORD fraction;
}
```

[1] Declare a C variable of type DATE_STRUCT
[2] Declare a C variable of type TIME_STRUCT
[3] Declare a C variable of type TIMESTAMP_STRUCT

PtrDataObj	The address of the column variable. Note that this is a void pointer because it may point to many different types of C variables.
SzDataObj	The size of the column in bytes.
KeyField	Indicates if this column is part of the primary index to the table.
Updateable	Indicates whether this column is updateable.
InsertAllowed	Indicates whether this column should be included in INSERT statements.
DerivedCol	Indicates whether this column is a base table column or a "derived column". A column may be derived from other base table columns, columns from join tables, SQL expressions, or any combination of all of them. Derived columns are not part of the base table. They will be ignored when CreateTable() is called. They are query only columns that cannot be INSERTed or UPDATEd. In fact, the Updateable and InsertAllowed settings are ignored and automatically set to FALSE for derived columns.
CbValue	Internal use only!!!

CidxDef

Overview

This structure is used when creating secondary indexes on your wxTable objects. Declare an array of CidxDef structures. The array should have as many elements as there are columns in your index. For each column you need to specify the column name and whether the column should be sorted in ascending or descending order.

Class Declaration

```
class CidxDef
{
public:
   char ColName[DB_MAX_COLUMN_NAME_LEN];
   Bool Ascending;
};          // CidxDef
```

Class Details

ColName	The column name.
Ascending	TRUE if ascending, FALSE if descending.

CcolInf

Overview

This structure is returned from wxDB::GetColumns to obtain column information about your tables.

Class Declaration

```
class CcolInf
{
public:
		char tableName[DB_MAX_TABLE_NAME_LEN+1];
		char colName[DB_MAX_COLUMN_NAME_LEN+1];
		int  sqlDataType;
};
```

Miscellaneous Functions

CloseDbConnections()

Declaration

void CloseDbConnections(void);

Comments

Called at the end of your application to close all of the open ODBC datasource connections.

FreeDbConnection()

Declaration

Bool FreeDbConnection(wxDB *pDb);

Arguments

pDb	Pointer to the wxDB object you'd like make available to another thread within the program.

Comments

Frees up the wxDB connection for use by another thread in the program. The connection is not closed, it is simply marked as "free" so it can be used by another portion of your application.

GetDataSource()

Declaration

```
Bool GetDataSource(HENV henv, char *Dsn, SWORD DsnMax,
                   char *DsDesc, SWORD DsDescMax,
                   UWORD direction = SQL_FETCH_NEXT);
```

Arguments

henv	ODBC environment handle
Dsn	Buffer allocated for the return data source name
DsnMax	Maximum length of the data source name buffer
DsDesc	Buffer allocated for the data source description
DsDescMax	Maximum length of the data source description buffer
direction	SQL_FETCH_FIRST or SQL_FETCH_NEXT

Comments

This function queries the ODBC driver manager for available ODBC data sources.

GetDbConnection()

Declaration

```
wxDB *GetDbConnection(DbStuff * pDbStuff);
```

Arguments

pDbStuff   A pointer to a structure of type DbStuff. The declaration of DbStuff is:

```
struct DbStuff
{
   HENV          Henv;
   char          Dsn[SQL_MAX_DSN_LENGTH];
   char          Uid[20];
   char          AuthStr[20];
};
```

This structure describes the datasource to ODBC. The Henv is the global ODBC environment handle. Dsn is the ODBC datasource name. Uid is the user id and AuthStr is the password.

Comments

Call GetDbConnection each time you need a new connection to the datasource. Multiple connections to a datasource(s) allows you to have multiple concurrent transactions within your application. It also allows for multi-database support within your application. The DbStuff argument describes the datasource to ODBC so it can connect to it. If the connection has already been made for this datasource and is currently free, then this routine will simply return a pointer to that connection. If no free connections are available, a new one will be created.

Example

```
        // Incomplete code sample
DbStuff ds;
ds.Henv = GlobalHenv;               // or whatever you're HENV variable is called
strcpy(ds.Dsn, "fp4:Sybase50:Sybase:32bit");
strcpy(ds.Uid, "fastpic");
strcpy(ds.AuthStr, "fastpic");
wxDB *sampleDB;
sampleDB = GetDbConnection(&ds);
.
.               // Use datasource connection
.
FreeDbConnection(sampleDB);
```

APPENDIX A – ODBC Reserved Words

List of Reserved Words

Following is a list of words reserved for use by ODBC function calls. For compatibility with drivers which support SQL grammar, programs should avoid using any of the keywords except in situations where they are used in a way the core SQL grammar specifies. The #define SQL_ODBC_KEYWORDS details a comma separated list of these keywords.

ABSOLUTE
ACTION
ADA
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BEGIN
BETWEEN
BIT
BIT_LENGTH
BOTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHAR_LENGTH
CHARACTER
CHARACTER_LENGTH
CHECK
CLOSE
COALESCE
COBOL
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DATE
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DISCONNECT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END-EXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FORTRAN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INCLUDE
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEADING
LEFT
LEVEL
LIKE
LOCAL
LOWER
MATCH
MAX
MIN
MINUTE
MODULE

MONTH
MUMPS
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PARTIAL
PASCAL
PLI
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
REFERENCES
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SEQUENCE
SESSION
SESSION_USER
SET
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCA
SQLCODE
SQLERROR
SQLSTATE
SQLWARNING
SUBSTRING
SUM
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRIM
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USING
VALUE
VALUES
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
YEAR