A wxDbTable instance provides re-usable access to rows of data in a table contained within the associated ODBC datasource

See the database classes overview for an introduction to using the ODBC classes.

Include files

<wx/dbtable.h>
<wx/db.h>

Members Helper classes and data structures

The following classes and structs are defined in dbtable.cpp/.h for use with the wxDbTable class.

```
* wxDbColDef
* wxDbColDataPtr
* wxDbIdxDef
```

Constants

wxDB_DEFAULT_CURSOR	Primary cursor normally used for cursor based operations.
wxDB_QUERY_ONLY	Used to indicate whether a table that is opened is for query only, or if insert/update/deletes will be performed on the table. Less overhead (cursors and memory) are allocated for query only tables, plus read access times are faster with some datasources.
wxDB_ROWID_LEN	[Oracle only] - Used when CanUpdateByRowID() is true. Optimizes updates so they are faster by updating on the Oracle-specific ROWID column rather than some other index.
wxDB_DISABLE_VIEW	Use to indicate when a database view should not be if a table is normally set up to use a view. [Currently unsupported.]

Members

wxDbTable::wxDbTable
wxDbTable::wxDbTable
wxDbTable::BuildDeleteStmt
wxDbTable::BuildSelectStmt
wxDbTable::BuildUpdateStmt
wxDbTable::BuildWhereClause
wxDbTable::CanSelectForUpdate
wxDbTable::CanUpdateByROWID
wxDbTable::ClearMemberVar

wxDbTable::ClearMemberVars
wxDbTable::CloseCursor
wxDbTable::Count
wxDbTable::CreateIndex
wxDbTable::CreateTable
wxDbTable::DB_STATUS
wxDbTable::Delete
wxDbTable::DeleteCursor
wxDbTable::DeleteMatching
wxDbTable::DeleteWhere
wxDbTable::DropIndex
wxDbTable::DropTable
wxDbTable::From
wxDbTable::GetColDefs
wxDbTable::GetCursor
wxDbTable::GetDb
wxDbTable::GetFirst
wxDbTable::GetFromClause
wxDbTable::GetLast
wxDbTable::GetNewCursor
wxDbTable::GetNext
wxDbTable::GetNumberOfColumns
wxDbTable::GetOrderByClause
wxDbTable::GetPrev
wxDbTable::GetQueryTableName
wxDbTable::GetRowNum
wxDbTable::GetTableName
wxDbTable::GetTablePath
wxDbTable::GetWhereClause
wxDbTable::Insert
wxDbTable::IsColNull
wxDbTable::IsCursorClosedOnCommit
wxDbTable::IsQueryOnly
wxDbTable::Open
wxDbTable::OrderBy
wxDbTable::Query
wxDbTable::QueryBySqlStmt
wxDbTable::QueryMatching
wxDbTable::QueryOnKeyFields
wxDbTable::Refresh
wxDbTable::SetColDefs
wxDbTable::SetCursor
wxDbTable::SetFromClause
wxDbTable::SetColNull

wxDbTable::SetOrderByClause
wxDbTable::SetQueryTimeout
wxDbTable::SetWhereClause
wxDbTable::Update
wxDbTable::UpdateWhere
wxDbTable::Where
wxDbTable::operator ++
wxDbTable::operator --

wxDbTable::wxDbTable

wxDbTable(wxDb *pwxDb, const wxString &tblName, const UWORD numColumns, const wxString &qryTblName = "", bool qryOnly = !wxDB_QUERY_ONLY, const wxString &tblPath = "")

Default constructor.

Parameters

pwxDb

Pointer to the wxDb instance to be used by this wxDbTable instance.

tblName

The name of the table in the RDBMS.

numColumns

The number of columns in the table. (Do NOT include the ROWID column in the count if using Oracle).

qryTblName

OPTIONAL. 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 wxDbTable object. Basing your queries on a view can provide a substantial performance increase in cases where your queries involve many tables with multiple joins. Default is "".

qryOnly

OPTIONAL. Indicates whether the table will be accessible for query purposes only, or should the table create the necessary cursors to be able to insert, update, and delete data from the table. Default is !wxDB_QUERY_ONLY.

tblPath

OPTIONAL. Some datasources (such as dBase) require a path to where the table is stored on the system. Default is "".

wxDbTable::wxDbTable

virtual ~wxDbTable()

Virtual default destructor.

wxDbTable::BuildDeleteStmt

void BuildDeleteStmt(wxString &pSqlStmt, int typeOfDel, const wxString &pWhereClause="")

Constructs the full SQL statement that can be used to delete all rows matching the criteria in the pWhereClause.

Parameters

pSqlStmt

Pointer to buffer 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

OPTIONAL. If the typeOfDel is DB_DEL_WHERE, then you must also pass in a SQL WHERE clause in this argument. Default is "".

Remarks

This member function constructs a SQL DELETE statement. This can be used for debugging purposes if you are having problems executing your SQL statement.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause are ignored by this function.

wxDbTable::BuildSelectStmt

void BuildSelectStmt(wxString &pSqlStmt, int typeOfSelect, bool distinct)

Constructs the full SQL statement that can be used to select all rows matching the criteria in the pWhereClause. This function is called internally in the wxDbTable class whenever the function wxDbTable::Query is called.

NOTE: Only the columns specified in wxDbTable::SetColDefs statements are included in the list of

columns returned by the SQL statement created by a call to this function.

Parameters

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

```
Whether to select distinct records only.
```

Remarks

This member function constructs a SQL SELECT statement. This can be used for debugging purposes if you are having problems executing your SQL statement.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause are ignored by this function.

wxDbTable::BuildUpdateStmt

void BuildUpdateStmt(wxString &pSqlStmt, int typeOfUpd, const wxString &pWhereClause="")

Constructs the full SQL statement that can be used to update all rows matching the criteria in the pWhereClause.

If typeOfUpdate is DB_UPD_KEYFIELDS, then the current values in the bound columns are used to determine which row(s) in the table are to be updated. The exception to this is when a datasource supports ROW IDs (Oracle). The ROW ID column is used for efficiency purposes when available.

NOTE: Only the columns specified in wxDbTable::SetColDefs statements are included in the list of columns updated by the SQL statement created by a call to this function. Any column definitions that were defined as being non-updateable will be excluded from the SQL UPDATE statement created by this function.

Parameters

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.

typeOfUpdate

The type of update statement being performed. Can be one of two values: DB_UPD_KEYFIELDS or DB_UPD_WHERE.

pWhereClause

OPTIONAL. If the typeOfUpdate is DB_UPD_WHERE, then you must also pass in a SQL WHERE clause in this argument. Default is "".

Remarks

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 are having problems executing your SQL statement.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause are ignored by this function.

wxDbTable::BuildWhereClause

void BuildWhereClause(wxString &pWhereClause, int typeOfWhere, const wxString &qualTableName="", bool useLikeComparison=false)

Constructs the portion of a SQL statement which would follow the word 'WHERE' in a SQL statement to be passed to the datasource. The returned string does NOT include the word 'WHERE'.

Parameters

pWhereClause

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.

typeOfWhere

The type of where clause to generate. Can be one of two values: DB_WHERE_KEYFIELDS or DB_WHERE_MATCHING.

qualTableName

OPTIONAL. Prepended to all base table column names. For use when a FROM clause has been specified with the wxDbTable::SetFromClause, to clarify which table a column name reference belongs to. Default is "".

useLikeComparison

```
            OPTIONAL. Should the constructed WHERE clause utilize the LIKE comparison operator. If
    false, then the '=' operator is used. Default is false.
```

Remarks

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 own SQL statements.

If using 'typeOfWhere' set to DB_WHERE_MATCHING, any bound columns currently containing a NULL value are not included in the WHERE clause's list of columns to use in the comparison.

wxDbTable::CanSelectForUpdate

bool CanSelectForUpdate()

Use this function to determine if the datasource supports SELECT ... FOR UPDATE. When the keywords "FOR UPDATE" are included as part of your SQL SELECT statement, all records retrieved (not just queried, but actually retrieved using wxDbTable::GetNext, etc) from the result set are locked.

Remarks

Not all datasources support the "FOR UPDATE" clause, so you must use this member function to determine if the datasource currently connected to supports this behavior or not before trying to select using "FOR UPDATE".

If the wxDbTable instance was created with the parameter wxDB_QUERY_ONLY, then this function will return false. For all known databases which do not support the FOR UPDATE clause, this function will return false also.

wxDbTable::CanUpdateByROWID

bool CanUpdateByROWID()

CURRENTLY ONLY POSSIBLE IF USING ORACLE.

--- CURRENTLY DISABLED FOR *ALL* DATASOURCES --- NOV 1 2000 - gt

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

Use of the ROWID feature is always handled by the class library except in the case of wxDbTable::QueryBySqlStmt. Since you are passing in the SQL SELECT statement, it is up to you to include the ROWID column in your query. If you do not, 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 wxDbTable object.

Remarks

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 datasource or not.

Example

```
// Incomplete code sample
wxDbTable parts;
.....
if (parts.CanUpdateByROWID())
{
    // Note that the ROWID column must always be the last column selected
    sqlStmt = "SELECT PART_NUM, PART_DESC, ROWID" FROM PARTS";
}
else
    sqlStmt = "SELECT PART_NUM, PART_DESC FROM PARTS";
```

wxDbTable::ClearMemberVar

void ClearMemberVar(UWORD colNumber, bool setToNull=false)

Same as wxDbTable::ClearMemberVars except that this function clears only the specified column of its values, and optionally sets the column to be a NULL column.

colNumber

Column number that is to be cleared. This number (between 0 and (numColumns-1)) is the index of the column definition created using the wxDbTable::SetColDefs function.

setToNull

OPTIONAL. Indicates whether the column should be flagged as being a NULL value stored in the bound memory variable. If true, then any value stored in the bound member variable is cleared. Default is false.

wxDbTable::ClearMemberVars

void ClearMemberVars(bool setToNull=false)

Initializes all bound columns of the wxDbTable instance to zero. In the case of a string, zero is copied to the first byte of the string.

setToNull

OPTIONAL. Indicates whether all columns should be flagged as having a NULL value stored in the bound memory variable. If true, then any value stored in the bound member variable is cleared. Default is false.

Remarks

This is useful before calling functions such as wxDbTable::QueryMatching or wxDbTable::DeleteMatching since these functions build their WHERE clauses from non-zero columns. To call either wxDbTable::QueryMatching or wxDbTable::DeleteMatching use this sequence:

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

wxDbTable::CloseCursor

bool CloseCursor(HSTMTcursor)

Closes the specified cursor associated with the wxDbTable object.

Parameters

cursor

The cursor to be closed.

Remarks

Typically handled internally by the ODBC class library, but may be used by the programmer if desired.

DO NOT CLOSE THE wxDB_DEFAULT_CURSOR!

wxDbTable::Count

ULONG Count(const wxString &args="*")

Returns the number of records which would be in the result set using the current query parameters specified in the WHERE and FROM clauses.

Parameters

args

OPTIONAL. This argument allows the use of the DISTINCT keyword against a column name to cause the returned count to only indicate the number of rows in the result set that have a unique value in the specified column. An example is shown below. Default is "*", meaning a count of the total number of rows matching is returned, regardless of uniqueness.

Remarks

This function can be called before or after an actual query to obtain the count of records in the result set. Count() uses its own cursor, so result set cursor positioning is not affected by calls to Count().

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause ARE used by this function.

Example

```
USERS TABLE


FIRST_NAME      LAST_NAME
----------      ----------
John            Doe
Richard         Smith
Michael         Jones
John            Carpenter


// Incomplete code sample
wxDbTable users;
.....
users.SetWhereClause("");


// This Count() will return 4, as there are four users listed above
// that match the query parameters
totalNumberOfUsers = users.Count();


// This Count() will return 3, as there are only 3 unique first names
// in the table above - John, Richard, Michael.
totalNumberOfUniqueFirstNames = users.Count("DISTINCT FIRST_NAME");
```

wxDbTable::CreateIndex

bool CreateIndex(const wxString &IndexName, bool unique, UWORD numIndexColumns, wxDbIdxDef *pIndexDefs, bool attemptDrop=true)

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.

Parameters

IndexName

```
Name of the Index. Name must be unique within the table space of the datasource.
```

unique

Indicates if this index is unique.

numIndexColumns

Number of columns in the index.

pIndexDefs

A pointer to an array wxDbIdxDef structures.

attemptDrop

OPTIONAL. Indicates if the function should try to execute a wxDbTable::DropIndex on the index name provided before trying to create the index name. Default is true.

Remarks

The first parameter, index name, must be unique and should be given a meaningful name. Common practice is to include the table name as a prefix in the index name (e.g. For table PARTS, you might want to call your index PARTS_Index1). This will allow you to easily view all of the indexes defined for a given table grouped together alphabetically.

The second parameter indicates if the index is unique or not. Uniqueness is enforced at the RDBMS level preventing rows which would have duplicate indexes from being inserted into the table when violating a unique index's uniqueness.

In the third parameter, specify how many columns are in your index. This number must match the number of columns defined in the 'pIndexDefs' parameter.

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

The fifth parameter is provided to handle the differences in datasources as to whether they will automatically overwrite existing indexes with the same name or not. Some datasources require that the existing index must be dropped first, so this is the default behavior.

Some datasources (MySQL, and possibly others) require columns which are to be part of an index to be defined as NOT NULL. When this function is called, if a column is not defined to be NOT NULL, a call to this function will modify the column definition to change any columns included in the index to be NOT NULL. In this situation, if a NULL value already exists in one of the columns that is being modified, creation of the index will fail.

PostGres is unable to handle index definitions which specify whether the index is ascending or descending, and defaults to the system default when the index is created.

It is not necessary to call wxDb::CommitTrans after executing this function.

Example

```
// Create a secondary index on the PARTS table
wxDbIdxDef  IndexDef[2];  // 2 columns make up the index


wxStrcpy(IndexDef[0].ColName, "PART_DESC");  // Column 1
IndexDef[0].Ascending = true;


wxStrcpy(IndexDef[1].ColName, "SERIAL_NO");  // Column 2
IndexDef[1].Ascending = false;


// Create a name for the index based on the table's name
wxString indexName;
indexName.Printf("%s_Index1",parts->GetTableName());
parts->CreateIndex(indexName, true, 2, IndexDef);
```

wxDbTable::CreateTable

bool CreateTable(bool attemptDrop=true)

Creates a table based on the definitions previously defined for this wxDbTable instance.

Parameters

attemptDrop

```
            OPTIONAL. Indicates whether the driver should attempt to drop the table before trying to
create it. Some datasources will not allow creation of a table if the table already exists in the
table space being used. Default is true.
```

Remarks

This function creates the table and primary index (if any) in the table space associated with the connected datasource. The owner of these objects will be the user id that was given when wxDb::Open was called. The objects will be created in the default schema/table space for that user.

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

It is not necessary to call wxDb::CommitTrans after executing this function.

wxDbTable::DB_STATUS

bool DB_STATUS()

Accessor function that returns the wxDb private member variable DB_STATUS for the database connection used by this instance of wxDbTable.

wxDbTable::Delete

bool Delete()

Deletes the row from the table indicated by the current cursor.

Remarks

Use wxDbTable::GetFirst, wxDbTable::GetLast, wxDbTable::GetNext or wxDbTable::GetPrev to position the cursor to a valid record. Once positioned on a record, call this function to delete the row from the table.

A wxDb::CommitTrans or wxDb::RollbackTrans must be called after use of this function to commit or rollback the deletion.

NOTE: Most datasources have a limited size "rollback" segment. This means that it is only possible to insert/update/delete a finite number of rows without performing a wxDb::CommitTrans or wxDb::RollbackTrans. Size of the rollback segment varies from database to database, and is user configurable in most databases. Therefore it is usually best to try to perform a commit or rollback at relatively small intervals when processing a larger number of actions that insert/update/delete rows in a table.

wxDbTable::DeleteCursor

bool DeleteCursor(HSTMT *hstmtDel)

Allows a program to delete a cursor.

Parameters

hstmtDel

Handle of the cursor to delete.

Remarks

For default cursors associated with the instance of wxDbTable, it is not necessary to specifically delete the cursors. This is automatically done in the wxDbTable destructor.

NOTE: If the cursor could not be deleted for some reason, an error is logged indicating the reason. Even if the cursor could not be deleted, the HSTMT that is passed in is deleted, and the pointer is set to NULL.

DO NOT DELETE THE wxDB_DEFAULT_CURSOR!

wxDbTable::DeleteMatching

bool DeleteMatching()

This member function allows you to delete records from your wxDbTable object by specifying the data in the columns to match on.

Remarks

To delete all users with a first name of "JOHN", do the following:

1. Clear all "columns" using wxDbTable::ClearMemberVars().
2. Set the FIRST_NAME column equal to "JOHN".
3. Call wxDbTable::DeleteMatching().

The WHERE clause is built by the ODBC class library based on all non-NULL columns. This allows deletion of records by matching on any column(s) in your wxDbTable instance, without having to write the SQL WHERE clause.

A wxDb::CommitTrans or wxDb::RollbackTrans must be called after use of this function to commit or rollback the deletion.

NOTE: Row(s) should be locked before deleting them to make sure they are not already in use. This can be achieved by calling wxDbTable::QueryMatching, and then retrieving the records, locking each as you go (assuming FOR UPDATE is allowed on the datasource). After the row(s) have been successfully locked, call this function.

NOTE: Most datasources have a limited "rollback" segment. This means that it is only possible to insert/update/delete a finite number of rows without performing a wxDb::CommitTrans or wxDb::RollbackTrans. Size of the rollback segment varies from database to database, and is user configurable in most databases. Therefore it is usually best to try to perform a commit or rollback at relatively small intervals when processing a larger number of actions that insert/update/delete rows in a table.

Example

```
// Incomplete code sample to delete all users with a first name
// of "JOHN"
users.ClearMemberVars();
wxStrcpy(users.FirstName,"JOHN");
users.DeleteMatching();
```

wxDbTable::DeleteWhere

bool DeleteWhere(const wxString &pWhereClause)

Deletes all rows from the table which match the criteria specified in the WHERE clause that is passed in.

Parameters

pWhereClause

SQL WHERE clause. This WHERE clause determines which records will be deleted from the table interfaced through the wxDbTable instance. The WHERE clause passed in must be compliant with the SQL 92 grammar. Do not include the keyword 'WHERE'

Remarks

This is the most powerful form of the wxDbTable delete functions. This function gives access to the full power of SQL. This function can be used to delete records by passing a valid SQL WHERE clause. Sophisticated deletions can be performed based on multiple criteria using the full functionality of the SQL language.

A wxDb::CommitTrans must be called after use of this function to commit the deletions.

Note: This function is limited to deleting records from the table associated with this wxDbTable object only. Deletions on joined tables is not possible.

NOTE: Most datasources have a limited size "rollback" segment. This means that it is only possible to insert/update/delete a finite number of rows without performing a wxDb::CommitTrans or wxDb::RollbackTrans. Size of the rollback segment varies from database to database, and is user configurable in most databases. Therefore it is usually best to try to perform a commit or rollback at relatively small intervals when processing a larger number of actions that insert/update/delete rows in a table.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause are ignored by this function.

Example

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

wxDbTable::DropIndex

bool DropIndex(const wxString &IndexName)

Allows an index on the associated table to be dropped (deleted) if the user login has sufficient privileges to do so.

Parameters

IndexName

Name of the index to be dropped.

Remarks

If the index specified in the 'IndexName' parameter does not exist, an error will be logged, and the function will return a result of false.

It is not necessary to call wxDb::CommitTrans after executing this function.

wxDbTable::DropTable

bool DropTable()

Deletes the associated table if the user has sufficient privileges to do so.

Remarks

This function returns true if the table does not exist, but only for supported databases (see wxDb::Dbms). If a datasource is not specifically supported, and this function is called, the function will return false.

Most datasources/ODBC drivers will delete any indexes associated with the table automatically, and others may not. Check the documentation for your database to determine the behavior.

It is not necessary to call wxDb::CommitTrans after executing this function.

wxDbTable::From

const wxString & From()

void From(const wxString &From)

Accessor function for the private class member wxDbTable::from. Can be used as a synonym for wxDbTable::GetFromClause (the first form of this function) or wxDbTable::SetFromClause (the second form of this function).

Parameters

From

A comma separated list of table names that are to be outer joined with the base table's columns so that the joined table's columns may be returned in the result set or used as a portion of a comparison with the base table's columns. NOTE that the base tables name must NOT be included in the FROM clause, as it is automatically included by the wxDbTable class in constructing query statements.

Return value

The first form of this function returns the current value of the wxDbTable member variable ::from.

The second form of the function has no return value, as it will always set the from clause successfully.

See also

wxDbTable::GetFromClause, wxDbTable::SetFromClause

wxDbTable::GetColDefs

wxDbColDef * GetColDefs()

Accessor function that returns a pointer to the array of column definitions that are bound to the columns that this wxDbTable instance is associated with.

To determine the number of elements pointed to by the returned wxDbColDef pointer, use the wxDbTable::GetNumberOfColumns function.

Remarks

These column definitions must not be manually redefined after they have been set.

wxDbTable::GetCursor

HSTMT GetCursor()

Returns the HSTMT value of the current cursor for this wxDbTable object.

Remarks

This function is typically used just before changing to use a different cursor so that after the program is finished using the other cursor, the current cursor can be set back to being the cursor in use.

See also

wxDbTable::SetCursor, wxDbTable::GetNewCursor

wxDbTable::GetDb

wxDb * GetDb()

Accessor function for the private member variable pDb which is a pointer to the datasource connection that this wxDbTable instance uses.

wxDbTable::GetFirst

bool GetFirst()

Retrieves the FIRST row in the record set as defined by the current query. Before retrieving records, a query must be performed using wxDbTable::Query, wxDbTable::QueryOnKeyFields, wxDbTable::QueryMatching or wxDbTable::QueryBySqlStmt.

Remarks

This function can only be used if the datasource connection used by the wxDbTable instance was created with FwdOnlyCursors set to false. If the connection does not allow backward scrolling cursors, this function will return false, and the data contained in the bound columns will be undefined.

See also

wxDb::IsFwdOnlyCursors

wxDbTable::GetFromClause

const wxString & GetFromClause()

Accessor function that returns the current FROM setting assigned with the wxDbTable::SetFromClause.

See also

wxDbTable::From

wxDbTable::GetLast

bool GetLast()

Retrieves the LAST row in the record set as defined by the current query. Before retrieving records, a query must be performed using wxDbTable::Query, wxDbTable::QueryOnKeyFields, wxDbTable::QueryMatching or wxDbTable::QueryBySqlStmt.

Remarks

This function can only be used if the datasource connection used by the wxDbTable instance was created with FwdOnlyCursors set to false. If the connection does not allow backward scrolling cursors, this function will return false, and the data contained in the bound columns will be undefined.

See also

wxDb::IsFwdOnlyCursors

wxDbTable::GetNewCursor

HSTMT * GetNewCursor(bool setCursor=false, bool bindColumns=true)

This function will create a new cursor that can be used to access the table being referenced by this wxDbTable instance, or to execute direct SQL commands on without affecting the cursors that are already defined and possibly positioned.

Parameters

setCursor

OPTIONAL. Should this new cursor be set to be the current cursor after successfully creating the new cursor. Default is false.

bindColumns

OPTIONAL. Should this new cursor be bound to all the memory variables that the default cursor is bound to. Default is true.

Remarks

This new cursor must be closed using wxDbTable::DeleteCursor by the calling program before the wxDbTable instance is deleted, or both memory and resource leaks will occur.

wxDbTable::GetNext

bool GetNext()

Retrieves the NEXT row in the record set after the current cursor position as defined by the current query. Before retrieving records, a query must be performed using wxDbTable::Query, wxDbTable::QueryOnKeyFields, wxDbTable::QueryMatching or wxDbTable::QueryBySqlStmt.

Return value

This function returns false when the current cursor has reached the end of the result set. When false is returned, data in the bound columns is undefined.

Remarks

This function works with both forward and backward scrolling cursors.

See also wxDbTable::++

wxDbTable::GetNumberOfColumns

UWORD GetNumberOfColumns()

Accessor function that returns the number of columns that are statically bound for access by the wxDbTable instance.

wxDbTable::GetOrderByClause

const wxString & GetOrderByClause()

Accessor function that returns the current ORDER BY setting assigned with the wxDbTable::SetOrderByClause.

See also

wxDbTable::OrderBy

wxDbTable::GetPrev

bool GetPrev()

Retrieves the PREVIOUS row in the record set before the current cursor position as defined by the current query. Before retrieving records, a query must be performed using wxDbTable::Query, wxDbTable::QueryOnKeyFields, wxDbTable::QueryMatching or wxDbTable::QueryBySqlStmt.

Return value

This function returns false when the current cursor has reached the beginning of the result set and there are now other rows prior to the cursors current position. When false is returned, data in the bound columns is undefined.

Remarks

This function can only be used if the datasource connection used by the wxDbTable instance was created with FwdOnlyCursors set to false. If the connection does not allow backward scrolling cursors, this function will return false, and the data contained in the bound columns will be undefined.

See also

wxDb::IsFwdOnlyCursors, wxDbTable::--

wxDbTable::GetQueryTableName

const wxString & GetQueryTableName()

Accessor function that returns the name of the table/view that was indicated as being the table/view to query against when this wxDbTable instance was created.

See also

wxDbTable constructor

wxDbTable::GetRowNum

UWORD GetRowNum()

Returns the ODBC row number for performing positioned updates and deletes.

Remarks

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.

Row number with some datasources/ODBC drivers is the position in the result set, while in others it may be a physical position in the database. Check your database documentation to find out which behavior is supported.

wxDbTable::GetTableName

const wxString & GetTableName()

Accessor function that returns the name of the table that was indicated as being the table that this wxDbTable instance was associated with.

wxDbTable::GetTablePath

const wxString & GetTablePath()

Accessor function that returns the path to the data table that was indicated during creation of this wxDbTable instance.

Remarks

Currently only applicable to dBase and MS-Access datasources.

wxDbTable::GetWhereClause

const wxString & GetWhereClause()

Accessor function that returns the current WHERE setting assigned with the wxDbTable::SetWhereClause

See also

wxDbTable::Where

wxDbTable::Insert

int Insert()

Inserts a new record into the table being referenced by this wxDbTable instance. The values in the member variables of the wxDbTable instance are inserted into the columns of the new row in the database.

Return value

```
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 (duplicate non-unique
                        index entry) is attempted.
```

Remarks

A wxDb::CommitTrans or wxDb::RollbackTrans must be called after use of this function to commit or rollback the insertion.

Example

```
// Incomplete code snippet
wxStrcpy(parts->PartName, "10");
wxStrcpy(parts->PartDesc, "Part #10");
parts->Qty = 1000;
RETCODE retcode = parts->Insert();
switch(retcode)
{
    case DB_SUCCESS:
        parts->GetDb()->CommitTrans();
        return(true);
    case DB_ERR_INTEGRITY_CONSTRAINT_VIOL:
        // Current data would result in a duplicate key
        // on one or more indexes that do not allow duplicates
        parts->GetDb()->RollbackTrans();
        return(false);
    default:
        // Insert failed for some unexpected reason
        parts->GetDb()->RollbackTrans();
        return(false);
}
```

wxDbTable::IsColNull

bool IsColNull(UWORD colNumber) const

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

Parameters

colNumber

```
The column number of the bound column as defined by the wxDbTable::SetColDefs calls which
defined the columns accessible to this wxDbTable instance.
```

Remarks

NULL column support is currently not fully implemented as of wxWidgets 2.4.

wxDbTable::IsCursorClosedOnCommit

bool IsCursorClosedOnCommit()

Accessor function to return information collected during the opening of the datasource connection that is used by this wxDbTable instance. The result returned by this function indicates whether an implicit closing of the cursor is done after a commit on the database connection.

Return value

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

Remarks

If more than one wxDbTable instance used the same database connection, all cursors which use the database connection are closed on the commit if this function indicates true.

wxDbTable::IsQueryOnly

bool IsQueryOnly()

Accessor function that returns a value indicating if this wxDbTable instance was created to allow only queries to be performed on the bound columns. If this function returns true, then no actions may be performed using this wxDbTable instance that would modify (insert/delete/update) the table's data.

wxDbTable::Open

bool Open(bool checkPrivileges=false, bool checkTableExists=true)

Every wxDbTable instance must be opened before it can be used. This function checks for the existence of the requested table, binds columns, creates required cursors, (insert/select and update if connection is not wxDB_QUERY_ONLY) and constructs the insert statement that is to be used for inserting data as a new row in the datasource.

NOTE: To retrieve data into an opened table, the of the table must be bound to the variables in the program via call(s) to wxDbTable::SetColDefs before calling Open().

See the database classes overview for an introduction to using the ODBC classes.

Parameters

checkPrivileges

Indicates whether the Open() function should check whether the current connected user has at least SELECT privileges to access the table to which they are trying to open. Default is false.

checkTableExists

Indicates whether the Open() function should check whether the table exists in the database or not before opening it. Default is true.

Remarks

If the function returns a false value due to the table not existing, a log entry is recorded for the datasource connection indicating the problem that was detected when checking for table existence. Note that it is usually best for the calling routine to check for the existence of the table and for sufficient user privileges to access the table in the mode (wxDB_QUERY_ONLY or !wxDB_QUERY_ONLY) before trying to open the table for the best possible explanation as to why a table cannot be opened.

Checking the user's privileges on a table can be quite time consuming during the open phase. With most applications, the programmer already knows that the user has sufficient privileges to access the table, so this check is normally not required.

For best performance, open the table, and then use the wxDb::TablePrivileges function to check the users privileges. Passing a schema to the TablePrivileges() function can significantly speed up the privileges checks.

See also

wxDb::TableExists, wxDb::TablePrivileges wxDbTable::SetColDefs

wxDbTable::OrderBy

const wxString & OrderBy()

void OrderBy(const wxString &OrderBy)

Accessor function for the private class member wxDbTable::orderBy. Can be used as a synonym for wxDbTable::GetOrderByClause (the first form of this function) or wxDbTable::SetOrderByClause (the second form of this function).

Parameters

OrderBy

A comma separated list of column names that indicate the alphabetized/numeric sorting sequence that the result set is to be returned in. If a FROM clause has also been specified, each column name specified in the ORDER BY clause should be prefaced with the table name to which the column belongs using DOT notation (TABLE_NAME.COLUMN_NAME).

Return value

The first form of this function returns the current value of the wxDbTable member variable ::orderBy.

The second form of the function has no return value.

See also

wxDbTable::GetOrderByClause, wxDbTable::SetFromClause

wxDbTable::Query

virtual bool Query(bool forUpdate=false, bool distinct=false)

Parameters

forUpdate

OPTIONAL. Gives you the option of locking records as they are retrieved. If the RDBMS is not capable of the FOR UPDATE clause, this argument is ignored. See wxDbTable::CanSelectForUpdate for additional information regarding this argument. Default is false.

distinct

OPTIONAL. Allows selection of only distinct values from the query (SELECT DISTINCT ... FROM ...). The notion of DISTINCT applies to all columns returned in the result set, not individual columns. Default is false.

Remarks

This function queries records from the datasource based on the three wxDbTable members: "where", "orderBy", and "from". Use wxDbTable::SetWhereClause to filter on records to be retrieved (e.g. All users with a first name of "JOHN"). Use wxDbTable::SetOrderByClause to change the sequence in which records are returned in the result set from the datasource (e.g. Ordered by LAST_NAME). Use wxDbTable::SetFromClause to allow outer joining of the base table (the one being associated with this instance of wxDbTable) with other tables which share a related field.

After each of these clauses are set/cleared, call wxDbTable::Query() to fetch the result set from the datasource.

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. Then to refresh the record set, simply call wxDbTable::Query() as frequently as needed.

Note that repeated calls to wxDbTable::Query() may tax the database server and make your application sluggish if done too frequently or unnecessarily.

The base table name is automatically prepended to the base column names in the event that the FROM clause has been set (is non-null) using wxDbTable::SetFromClause.

The cursor for the result set is positioned before the first record in the result set after the query. To retrieve the first record, call either wxDbTable::GetFirst (only if backward scrolling cursors are available) or wxDbTable::GetNext. Typically, no data from the result set is returned to the client driver until a request such as wxDbTable::GetNext is performed, so network traffic and database load are not overwhelmed transmitting data until the data is actually requested by the client. This behavior is solely dependent on the ODBC driver though, so refer to the ODBC driver's reference material for information on its behaviors.

Values in the bound columns' memory variables are undefined after executing a call to this function and remain that way until a row in the result set is requested to be returned.

The wxDbTable::Query() function is defined as "virtual" so that it may be overridden for application specific purposes.

Be sure to set the wxDbTable's "where", "orderBy", and "from" member variables to "" if they are not to be used in the query. Otherwise, the results returned may have unexpected results (or no results) due to improper or incorrect query parameters constructed from the uninitialized clauses.

Example

```
// Incomplete code sample
parts->SetWhereClause("DESCRIPTION = 'FOOD'");
parts->SetOrderByClause("EXPIRATION_DATE");
parts->SetFromClause("");
// Query the records based on the where, orderBy and from clauses
// specified above
parts->Query();
// Display all records queried
while(parts->GetNext())
    dispPart(parts);  // user defined function
```

wxDbTable::QueryBySqlStmt

bool QueryBySqlStmt(const wxString &pSqlStmt)

Performs a query against the datasource by accepting and passing verbatim the SQL SELECT statement passed to the function.

Parameters

pSqlStmt

Pointer to the SQL SELECT statement to be executed.

Remarks

This is the most powerful form of the query functions available. This member function allows a programmer to write their own custom SQL SELECT statement for requesting data from the datasource. This gives the programmer access to the full power of SQL for performing operations such as scalar functions, aggregate functions, table joins, and sub-queries, as well as datasource specific function calls.

The requirements of the SELECT statement are the following:

1. Must return the correct number of columns. In the derived wxDbTable constructor, it is specified how many columns are in the wxDbTable object. The SELECT statement must return exactly that many columns.
2. The columns must be returned in the same sequence as specified when defining the bounds columns wxDbTable::SetColDefs, and the columns returned must be of the proper data type. For example, if column 3 is defined in the wxDbTable bound column definitions to be a float, the SELECT statement must return a float for column 3 (e.g. PRICE * 1.10 to increase the price by 10
3. The ROWID can be included in your SELECT statement as the last column selected, if the datasource supports it. Use wxDbTable::CanUpdateByROWID() to determine if the ROWID can be selected from the datasource. If it can, much better performance can be achieved on updates and deletes by including the ROWID in the SELECT statement.

Even though data can be selected from multiple tables (joins) in your select statement, only the base table associated with this wxDbTable object is automatically updated through the ODBC class library. Data from multiple tables can be selected for display purposes however. Include columns in the wxDbTable object and mark them as non-updateable (See wxDbColDef for details). This way columns can be selected and displayed from other tables, but only the base table will be updated automatically when performed through the wxDbTable::Update function after using this type of query. To update tables other than the base table, use the wxDbTable::Update function passing a SQL statement.

After this function has been called, the cursor is positioned before the first record in the record set. To retrieve the first record, call either wxDbTable::GetFirst or wxDbTable::GetNext.

Example

```
 // Incomplete code samples
wxString sqlStmt;
 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->GetNext())
     dispPart(&parts);


 Example SQL statements
 ----------------------


 // Table Join returning 3 columns
 SELECT PART_NUM, 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_NUM, substring(part_desc, 1, 10), qty_on_hand + 1, ROWID
    from parts
    where warehouse = 10
    order by PART_NUM desc           // descending order


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

wxDbTable::QueryMatching

virtual bool QueryMatching(bool forUpdate=false, bool distinct=false)

QueryMatching allows querying of records from the table associated with the wxDbTable object by matching "columns" to values.

For example: To query the datasource for the row with a PART_NUMBER column value of "32", clear all column variables of the wxDbTable object, set the PartNumber variable that is bound to the PART_NUMBER column in the wxDbTable object to "32", and then call wxDbTable::QueryMatching().

Parameters

forUpdate

```
            OPTIONAL. 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
wxDbTable::CanSelectForUpdate for additional information regarding this argument. Default is false.
```

distinct

```
            OPTIONAL. Allows selection of only distinct values from the query (SELECT DISTINCT ...
FROM ...). The notion of DISTINCT applies to all columns returned in the result set, not individual
columns. Default is false.
```

Remarks

The SQL WHERE clause is built by the ODBC class library based on all non-zero/non-NULL columns in your wxDbTable object. Matches can be on one, many or all of the wxDbTable's columns. The base table name is prepended to the column names in the event that the wxDbTable's FROM clause is non-null.

This function cannot be used to perform queries which will check for columns that are 0 or NULL, as the automatically constructed WHERE clause only will contain comparisons on column member variables that are non-zero/non-NULL.

The primary difference between this function and wxDbTable::QueryOnKeyFields is that this function can query on any column(s) in the wxDbTable object. 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 the query is performed. To retrieve the first record, the program must call either wxDbTable::GetFirst or wxDbTable::GetNext.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause are ignored by this function.

Example

```
// Incomplete code sample
parts->ClearMemberVars();           // Set all columns to zero
wxStrcpy(parts->PartNumber,"32");   // Set columns to query on
parts->OnHold = true;
parts->QueryMatching();             // Query
// Display all records queried
while(parts->GetNext())
    dispPart(parts);  // Some application defined function
```

wxDbTable::QueryOnKeyFields

bool QueryOnKeyFields(bool forUpdate=false, bool distinct=false)

QueryOnKeyFields provides an easy mechanism to query records in the table associated with the wxDbTable 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 wxDbTable's FROM clause is non-null.

Parameters

forUpdate

OPTIONAL. 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 wxDbTable::CanSelectForUpdate for additional information regarding this argument. Default is false.

distinct

OPTIONAL. Allows selection of only distinct values from the query (SELECT DISTINCT ... FROM ...). The notion of DISTINCT applies to all columns returned in the result set, not individual columns. Default is false.

Remarks

The cursor is positioned before the first record in the record set after the query is performed. To retrieve the first record, the program must call either wxDbTable::GetFirst or wxDbTable::GetNext.

WHERE and FROM clauses specified using wxDbTable::SetWhereClause and wxDbTable::SetFromClause are ignored by this function.

Example

```
// Incomplete code sample
wxStrcpy(parts->PartNumber, "32");
parts->QueryOnKeyFields();
// Display all records queried
while(parts->GetNext())
    dispPart(parts);  // Some application defined function
```

wxDbTable::Refresh

bool Refresh()

この関数はバインドされたカラムをメモリ変数に再度読み込み、それらをディスクに格納されている現在の値に設定する。

This function re-reads the bound columns into the memory variables, setting them to the current values stored on the disk.

この関数を読んでも、カーソル位置と結果集合は変わらない（１つの例外は、リフレッシュされるレコードが他のユーザやトランザクションによって削除される場合である。大部分のデータソースにおいて、この場合のデフォルトの動作は、たとえデータベースから削除されていたとしても、結果集合について元々クエリした値を返す。しかし、これはデータソースに依存し、この動作をあてにする前に検証しておかなければならない)。

The cursor position and result set are unaffected by calls to this function. (The one exception is in the case where the record to be refreshed has been deleted by some other user or transaction since it was originally retrieved as part of the result set. For most datasources, the default behavior in this situation is to return the value that was originally queried for the result set, even though it has been deleted from the database. But this is datasource dependent, and should be tested before relying on this behavior.)

plugin::pdf::PDFParser=HASH(0x53e16c) 注意
この関数は、テーブルが唯一のプライマリインデックスを持っている場合にのみ動作することが保証されている。そうでなければ、１つ以上のレコードがフェッチされ、リフレッシュされるべき正しいレコードを保証できなくなる。データベース内の現在のデータを反映するために、テーブルのカラムがリフレッシュされる。

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.

wxDbTable::SetColDefs

bool SetColDefs(UWORD index, const wxString &fieldName, int dataType, void *pData, SWORD cType, int size, bool keyField = false, bool updateable = true, bool insertAllowed = true, bool

derivedColumn = false)

wxDbColDataPtr * SetColDefs(wxDbColInf *colInfs, UWORD numCols)

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

index

カラム数。（0 から n-1。ここで、n は wxDbTable コンストラクタが呼ばれたときに、この wxDbTable インスタンスの定義をする際に指定されたカラム数である。）

fieldName

関連付けられるデータテーブル内のカラム名。

dataType

論理的なデータ種別。有効な論理種別は以下の通り。

```
DB_DATA_TYPE_VARCHAR        : 文字列
DB_DATA_TYPE_INTEGER        : 非浮動小数点数
DB_DATA_TYPE_FLOAT          : 浮動小数点数
DB_DATA_TYPE_DATE           : 日付
```

pData

データ行がデータソースから返される際に、カラムの値を保持するためのデータオブジェクトへのポインタ。

cType

SQL C 種別。データを SQL 表現し、pData に格納する際のデータ種別。他にも利用可能な種別があるが、最も共通なものは以下の通り。

```
SQL_C_CHAR      // 文字列。できれば、SQL_C_WXCHAR を使用すべき。
SQL_C_WXCHAR    // 文字列。unicode でも非 unicode でも意識せずに使用できる。
SQL_C_LONG
SQL_C_ULONG
SQL_C_SHORT
SQL_C_USHORT
SQL_C_FLOAT
SQL_C_DOUBLE
SQL_C_NUMERIC
SQL_C_TIMESTAMP


SQL_C_BOOLEAN   // db.h に定義されている
SQL_C_ENUM      // db.h に定義されている
```

size

pData オブジェクトの最大バイト数。

keyField

オプション。このカラムがプライマリインデックスの一部かどうかを示す。デフォルトは false。

updateable

オプション。このカラムの更新が許されるかどうか。デフォルトは true。

insertAllowed

オプション。このカラムへの挿入が許されるかどうか。デフォルトは true。

derivedColumn

オプション。派生したカラム（クエリ専用で、ベーステーブルのカラムではない）かどうか。デフォルトは false。

colInfs

numCols 個のカラム定義を作成するために必要な全ての情報を含む、wxDbColInf インスタンスの配列へのポインタ。

numCols

colInfs で示される wxDbColInf 型変数の要素数。wxDbColInf により作成されたカラム定義である。

index

Column number (0 to n-1, where n is the number of columns specified as being defined for this wxDbTable instance when the wxDbTable constructor was called.

fieldName

Column name from the associated data table.

dataType

Logical data type. Valid logical types include:

DB_DATA_TYPE_VARCHAR : strings
DB_DATA_TYPE_INTEGER : non-floating point numbers
DB_DATA_TYPE_FLOAT : floating point numbers
DB_DATA_TYPE_DATE : dates

pData

Pointer to the data object that will hold the column's value when a row of data is returned from the datasource.

cType

SQL C Type. This defines the data type that the SQL representation of the data is converted to to be stored in pData. Other valid types are available also, but these are the most common ones:

SQL_C_CHAR // string - deprecated: use SQL_C_WXCHAR
SQL_C_WXCHAR // string - Used transparently in unicode or non-unicode builds
SQL_C_LONG
SQL_C_ULONG
SQL_C_SHORT
SQL_C_USHORT
SQL_C_FLOAT
SQL_C_DOUBLE
SQL_C_NUMERIC
SQL_C_TIMESTAMP
SQL_C_BOOLEAN // defined in db.h
SQL_C_ENUM // defined in db.h

size

Maximum size in bytes of the pData object.

keyField

OPTIONAL. Indicates if this column is part of the primary index. Default is false.

updateable

OPTIONAL. Are updates allowed on this column? Default is true.

insertAllowed

OPTIONAL. Inserts allowed on this column? Default is true.

derivedColumn

OPTIONAL. Is this a derived column (non-base table column for query only)? Default is false.

colInfs

Pointer to an array of wxDbColInf instances which contains all the information necessary to create numCols column definitions.

numCols

Number of elements of wxDbColInf type that are pointed to by colInfs, which are to have column definitions created from them.

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

pData が文字列の場合、'size' には pData に NULL 終端を含められるだけのバイト数を指定し、そのスペースを確保しなければならない。

If pData is to hold a string of characters, be sure to include enough space for the NULL terminator in pData and in the byte count of size.

この関数のはじめの書式を使用すると、カラム定義を作成できなかった場合には、false が返る。指定されたカラムのインデックスが wxDbTable インスタンスで定義されたカラム数を超えると、assert を投げ。（デバッグビルドの場合は）ログに出力し、false を返す。

Using the first form of this function, if the column definition is not able to be created, a value of false is returned. If the specified index of the column exceeds the number of columns defined in the wxDbTable instance, an assert is thrown and logged (in debug builds) and a false is returned.

２つ目の書式でカラム定義に失敗すると、NULL が返る。

A failure to create the column definition in the second form results in a value of NULL being returned.

どちらの書式も、自身の wxDbTable オブジェクトでカラムを定義するためのショートカットが用意されている。wxDbTable オブジェクトでカラムを記述する際には、任意に派生された wxDbTable コンストラクタの中でこの関数を使用すれば良い。

Both forms of this function provide a shortcut for defining the columns in your wxDbTable object. Use this function in any derived wxDbTable constructor when describing the column/columns in the wxDbTable object.

カラム定義についてデータソースにクエリするために wxDb::GetColums 関数が使用される場合には、カラム定義が既に wxDbColInfo に格納されていなければならないため、その前にこの関数の２つ目の書式が使用される。wxDb::GetColums を使用した後に個の関数を使用する使用例は、データテーブルが１つのデータソースに存在するときに、そのテーブルのカラム定義を他のデータソースやテーブルにコピーする場合である。

The second form of this function is primarily used when the wxDb::GetColumns function was used to query the datasource for the column definitions, so that the column definitions are already stored in wxDbColInf form. One example use of using wxDb::GetColumns then using this function is if a data table existed in one datasource, and the table's column definitions were to be copied over to another datasource or table.

plugin::pdf::PDFParser=HASH(0x53e16c) 使用例

```
// この関数を使用しない冗長な記述
wxStrcpy(colDefs[0].ColName, "PART_NUM");
colDefs[0].DbDataType   = DB_DATA_TYPE_VARCHAR;
colDefs[0].PtrDataObj   = PartNumber;
colDefs[0].SqlCtype     = SQL_C_WXCHAR;
colDefs[0].SzDataObj    = PART_NUMBER_LEN;
colDefs[0].KeyField     = true;
colDefs[0].Updateable   = false;
colDefs[0].InsertAllowed= true;
colDefs[0].DerivedCol   = false;
```

```
// この関数を使用したショートカット
SetColDefs(0, "PART_NUM", DB_DATA_TYPE_VARCHAR, PartNumber,
           SQL_C_WXCHAR, PART_NUMBER_LEN, true, false, true, false);
```

wxDbTable::SetCursor

void SetCursor(HSTMT *hstmtActivate = (void **) wxDB_DEFAULT_CURSOR)

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

hstmtActivate

オプション。カレントカーソルに設定するカーソルへのポインタ。カーソルハンドルを渡さないと、wxDbTable インスタンスが始めに作られたときに作成された、wxDbTable のデフォルトの（オリジナルの）カーソルに戻すことができる。デフォルトは wxDB_DEFAULT_CURSOR である。

OPTIONAL. Pointer to the cursor that is to become the current cursor. Passing no cursor handle will reset the cursor back to the wxDbTable's default (original) cursor that was created when the wxDbTable instance was first created. Default is wxDB_DEFAULT_CURSOR.

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

カーソルを入れ替える際、wxDbTable オブジェクトのメンバ変数は、カレントカーソルが示している列のカラム値に自動的に書き換えられる。もしカーソルが何も示していない場合、メンバ変数内のデータは未定義である。
この関数を呼ぶ前に使用されていたカーソルに戻すための唯一の方法は、この関数を呼ぶ前に wxDbTable::GetCursor を使用してカレントカーソルの HSTMT をプログラム的に取得し、そのカーソルへのポインタを保存しておくことである。

When swapping between cursors, the member variables of the wxDbTable object are automatically refreshed with the column values of the row that the current cursor is positioned at (if any). If the cursor is not positioned, then the data in member variables is undefined.

The only way to return back to the cursor that was in use before this function was called is to programmatically determine the current cursor's HSTMT BEFORE calling this function using wxDbTable::GetCursor and saving a pointer to that cursor.

plugin::pdf::PDFParser=HASH(0x53e16c) 参考

wxDbTable::GetNewCursor, wxDbTable::GetCursor, wxDbTable::SetCursor

wxDbTable::SetFromClause

void SetFromClause(const wxString &From)

外部のテーブルに含まれるカラムにアクセスするため、wxDbTable のベーステーブルと外部結合される外部テーブルを示す、プライベートなクラスメンバ wxDbTable::from を設定するためのアクセス関数。

Accessor function for setting the private class member wxDbTable::from that indicates what other tables should be outer joined with the wxDbTable's base table for access to the columns in those other tables.

Synonym to this function is one form of wxDbTable::From

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

From

結合されたテーブルのカラムが結果集合に返されるよう、また、そのカラムがベーステーブルのカラムと比較できるよう、ベーステーブルのカラムと外部結合されるテーブル名のリストをカンマ区切りしたリスト。wxDbTable クラスはクエリ文を作成する際、自動的にベーステーブルを FROM 節に含めるため、FROM 節にはベーステーブル名を含んではならない。

A comma separated list of table names that are to be outer joined with the base table's columns so that the joined table's columns may be returned in the result set or used as a portion of a comparison with the base table's columns. NOTE that the base tables name must NOT be included in the FROM clause, as it is automatically included by the wxDbTable class in constructing query statements.

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

複数のテーブルのレコードを外部結合するために、wxDbTable::Query と wxDbTable::Count メンバ関数から呼ばれる。

FROM 節を使用する際、キーワード "FROM" を含んではならない。

クエリを実行するときに FROM 節を使用する際、ベーステーブルのカラム名なのか、外部結合されたテーブルのカラム名なのか、どちらのカラムなのかをデータソースが知りうるよう、対応する WHERE 節を含めなければならない。

Used by the wxDbTable::Query and wxDbTable::Count member functions to allow outer joining of records from multiple tables.

Do not include the keyword "FROM" when setting the FROM clause.

If using the FROM clause when performing a query, be certain to include in the corresponding WHERE clause a comparison of a column from either the base table or one of the other joined tables to each other joined table to ensure the datasource knows on which column values the tables should be joined on.

plugin::pdf::PDFParser=HASH(0x53e16c) 例

```
...
// ベーステーブルは "LOCATION" テーブルであり、
// "PART" テーブルが共通フィールド "PART_NUMBER" を介して
// 外部結合されている。
location->SetWhereClause("LOCATION.PART_NUMBER = PART.PART_NUMBER")
location->SetFromClause("PART");
...
```

plugin::pdf::PDFParser=HASH(0x53e16c) 参考

wxDbTable::From, wxDbTable::GetFromClause

wxDbTable::SetColNull

bool SetColNull(UWORD colNumber, bool set=true)

bool SetColNull(const wxString &colName, bool set=true)

どちらの書式も、この wxDbTable オブジェクトに関連付けられたテーブル内のカラムを象徴するメンバ変数を NULL に設定する。

１つ目の書式は、wxDbTable インスタンスを作成する際に使用されたカラム定義内のインデックスでカラムを指定する。一方、２つ目の書式は、指定された実際のカラム名を指定する。

Both forms of this function allow a member variable representing a column in the table associated with this wxDbTable object to be set to NULL.

The first form allows the column to be set by the index into the column definitions used to create the wxDbTable instance, while the second allows the actual column name to be specified.

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

colNumber

始めにこの wxDbTable オブジェクトを定義したときに使用したカラム定義のインデックス

Index into the column definitions used when first defining this wxDbTable object.

colName

NULL にしたい、実際のデータテーブルカラム名

Actual data table column name that is to be set to NULL.

set

カラムを NULL にするかどうか。true を渡すとカラムは NULL になり、false を指定すると、NULL にしない。デフォルトは true。

Whether the column is set to NULL or not. Passing true sets the column to NULL, passing false sets the column to be non-NULL. Default is true.

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

この関数はデータベースを更新しない。メモリ内のメンバ変数への操作だけである。この値をディスクに保存する場合は、挿入関数や更新関数を使用する。

No database updates are done by this function. It only operates on the member variables in memory. Use and insert or update function to store this value to disk.

wxDbTable::SetOrderByClause

void SetOrderByClause(const wxString &OrderBy)

クエリの結果集合に含まれる行の順番を決定する、プライベートなクラスメンバ wxDbTable::orderBy を指定するためのアクセス関数。
関数 wxDbTable::OrderBy の１つの書式と同じ意味である。

Accessor function for setting the private class member wxDbTable::orderBy which determines sequence/ordering of the rows returned in the result set of a query.

A synonym to this function is one form of the function wxDbTable::OrderBy

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

OrderBy

返される結果集合をアルファベット順に並び替えることを示すカラム名をカンマ区切りしたリスト。FROM 節も指定する場合、ORDER BY 節で指定される各カラム名は、ドット表記を使用して、カラムが属するテーブル名に続いて記述される（TABLE_NAME.COLUMN_NAME）。

A comma separated list of column names that indicate the alphabetized sorting sequence that the result set is to be returned in. If a FROM clause has also been specified, each column name specified in the ORDER BY clause should be prefaced with the table name to which the column belongs using DOT notation (TABLE_NAME.COLUMN_NAME).

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

ORDER BY 節を設定する際、キーワード "ORDER BY" を含めてはならない。

Do not include the keywords "ORDER BY" when setting the ORDER BY clause.

plugin::pdf::PDFParser=HASH(0x53e16c) 例

```
...
parts->SetOrderByClause("PART_DESCRIP, QUANTITY");
...

...
location->SetOrderByClause("LOCATION.POSITION, PART.PART_NUMBER);
...
```

plugin::pdf::PDFParser=HASH(0x53e16c) 参考

wxDbTable::OrderBy, wxDbTable::GetOrderByClause

wxDbTable::SetQueryTimeout

bool SetQueryTimeout(UDWORD nSeconds)

クエリのためのタイムアウト時間を設定する。

Allows a time period to be set as the timeout period for queries.

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

nSeconds

クエリが完了するまでのタイムアウト時間（秒数）

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

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

今のところ、Oracle も Access もこの関数をサポートしていない。この関数が正常に動作するかどうかに依存する前に、他のデータベースでサポートされているかどうか評価する必要がある。

Neither Oracle or Access support this function as of yet. Other databases should be evaluated for support before depending on this function working correctly.

wxDbTable::SetWhereClause

void SetWhereClause(const wxString &Where)

データソースによって結果集合に返される行を決定するための、プライベートなクラスメンバ wxDbTable::where を設定するためのアクセス関数。
この関数は、wxDbTable::Where 関数の１つの書式と同じ意味である。

Accessor function for setting the private class member wxDbTable::where that determines which rows are returned in the result set by the datasource.
A synonym to this function is one form of the function wxDbTable::Where

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

Where

SQL "where" 節。この節には、正しい where 節を含む SQL 言語を含めることができる。FROM 節も指定されている場合、ORDER BY 節で指定された各カラム名は、ドット表記を使用することにより、カラムが属するテーブル名に続いて記述される（TABLE_NAME.COLUMN_NAME）。

SQL "where" clause. This clause can contain any SQL language that is legal in standard where clauses. If a FROM clause has also been specified, each column name specified in the ORDER BY clause should be prefaced with the table name to which the column belongs using DOT notation (TABLE_NAME.COLUMN_NAME).

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

WHERE 節を設定する際、キーワード "WHERE" を含んではならない。

Do not include the keywords "WHERE" when setting the WHERE clause.

plugin::pdf::PDFParser=HASH(0x53e16c) 例

```
...
// 単純な where 節
parts->SetWhereClause("PART_NUMBER = '32'");
...
// 任意の比較操作
parts->SetWhereClause("PART_DESCRIP LIKE 'HAMMER%'");
...
// 関数呼び出しを含む、複数条件
parts->Where("QTY > 0 AND {fn UCASE(PART_DESCRIP)} LIKE '%DRILL%'");
...
// 引数と、複数の論理の組み合わせ
parts->Where("((QTY > 10) OR (ON_ORDER > 0)) AND ON_HOLD = 0");
...
// このクエリは、共通なフィールド PART_NUMBER を持つ
// PART と LOCATION テーブルを結合して（FROM 節で要求された）外部結合を使用する
parts->Where("PART.ON_HOLD = 0 AND \
              PART.PART_NUMBER = LOCATION.PART_NUMBER AND \
              LOCATION.PART_NUMBER > 0");
```

plugin::pdf::PDFParser=HASH(0x53e16c) 参考

wxDbTable::Where, wxDbTable::GetWhereClause

wxDbTable::Update

bool Update()

bool Update(const wxString &pSqlStmt)

始めの書式は、現在のカーソルが現在示している行を、カラムにバインドされたメモリ変数内の変数で更新する。更新を実行するための実際の SQL 文は、ODBC クラスにより自動的に作成され、実行される。

The first form of this function will update the row that the current cursor is currently positioned at with the values in the memory variables that are bound to the columns. The actual SQL statement to perform the update is automatically created by the ODBC class, and then executed.

２つ目の書式は、データベース内のレコードを更新するための SQL 文を介したフルアクセスが可能である。正確な SQL UPDATE 文を書き、実行するためにこの関数に渡さなければならない。SQL の方言を最大限に使用することにより、より高度な更新を実行できる。SQL 文は、更新を実行するためにドライバ、データソースに要求された正確なシンタックスである必要である。通常、以下のように使用される。

```
UPDATE tablename SET col1=X, col2=Y, ... where ...
```

The second form of the function allows full access through SQL statements for updating records in the database. Write any valid SQL UPDATE statement and submit it to this function for execution. Sophisticated updates can be performed using the full power of the SQL dialect. The full SQL statement must have the exact syntax required by the driver/datasource for performing the update. This usually is in the form of:

UPDATE tablename SET col1=X, col2=Y, ... where ...

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

pSqlStmt
実行する SQL UPDATE 文へのポインタ

Pointer to SQL UPDATE statement to be executed.

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

更新をコミット、ロールバックするために、この関数を使用した後、wxDb::CommitTrans または wxDb::RollbackTrans を呼ばなければならない。

A wxDb::CommitTrans or wxDb::RollbackTrans must be called after use of this function to commit or rollback the update.

plugin::pdf::PDFParser=HASH(0x53e16c) 例

```
wxString sqlStmt;
sqlStmt = "update PART set QTY = 0 where PART_NUMBER = '32'";
```

wxDbTable::UpdateWhere

bool UpdateWhere(const wxString &pWhereClause)

wxDbTable オブジェクトのベーステーブルを更新する。この時、pWhereClause で指定された判断基準に合致する行だけが更新される。
この wxDbTable インスタンスのメンバ変数にバインドされた、更新可能であると定義された全てのカラムに対し、現在メモリ変数に保持している情報で更新する。

Performs updates to the base table of the wxDbTable object, updating only the rows which match the criteria specified in the pWhereClause.
All columns that are bound to member variables for this wxDbTable instance that were defined with the "updateable" parameter set to true will be updated with the information currently held in the memory variable.

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

pWhereClause
正確な SQL WHERE 節へのポインタ。キーワード 'WHERE' を含んではならない。

Pointer to a valid SQL WHERE clause. Do not include the keyword 'WHERE'.

plugin::pdf::PDFParser=HASH(0x53e16c) 注意

この関数を使用してインデックスの一部であるカラムを更新する場合には、ユニークなキー制約を破らないように使用しなければならない。
更新のコミットやロールバックのために、この関数を使用した後、wxDb::CommitTrans または wxDb::RollbackTrans を呼ばなければならない。

Care should be used when updating columns that are part of indexes with this function so as not to violate an unique key constraints.
A wxDb::CommitTrans or wxDb::RollbackTrans must be called after use of this function to commit or rollback the update(s).

wxDbTable::Where

const wxString & Where()

void Where(const wxString& Where)

プライベートなクラスメンバ wxDbTable::where へのアクセス関数。このテーブルインスタンスに対する、現在の where 節を返すための wxDbTable::GetWhereClause（始めの書式）や、where 節を設定するための wxDbTable::SetWhereClause（２つ目の書式）と同じように使うことができる。

plugin::pdf::PDFParser=HASH(0x53e16c) 引数

Where
正当な SQL の WHERE 節。キーワード 'WHERE' を含んではならない。

plugin::pdf::PDFParser=HASH(0x53e16c) 戻り値

始めの書式の関数は、wxDbTable メンバ変数 ::where の現在の値を返す。

２つ目の書式の関数は、戻り値を持たず、成功したときには常に where 節を設定する。

plugin::pdf::PDFParser=HASH(0x53e16c) 参考

wxDbTable::GetWhereClause, wxDbTable::SetWhereClause

wxDbTable::operator ++

bool operator ++()

wxDbTable::GetNext と同じものである。

plugin::pdf::PDFParser=HASH(0x53e16c) 参考
wxDbTable::GetNext

wxDbTable::operator --

bool operator --()

wxDbTable::GetPrev と同じものである。

plugin::pdf::PDFParser=HASH(0x53e16c) 参考
wxDbTable::GetPrev