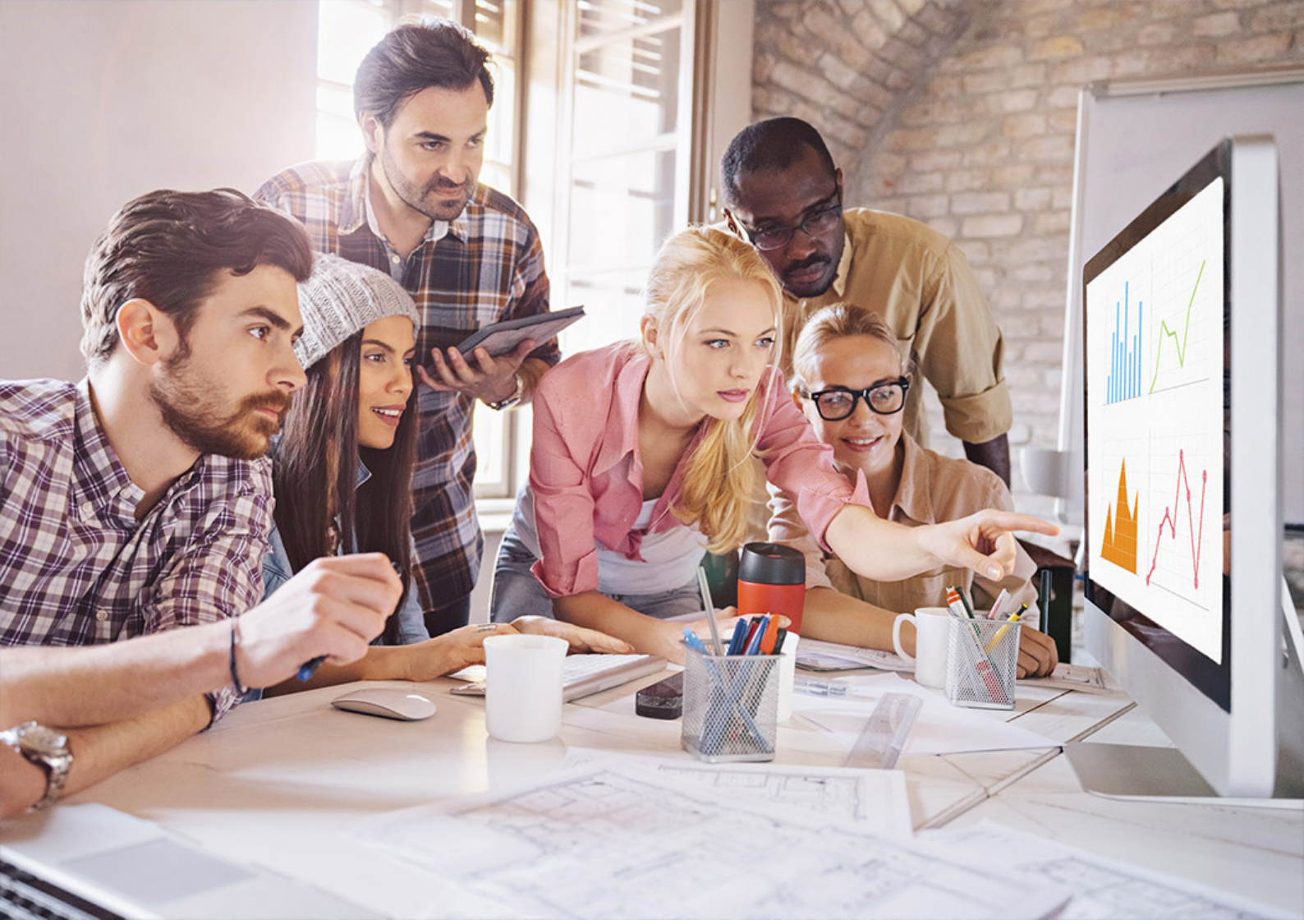

PUBLIC
SQL Anywhere - UltraLite
Document Version: 17.01.0 – 2021-10-15

UltraLite - C++ API Reference

THE BEST RUN SAP

Content

1 UltraLite C++ API Reference

UltraLite C++ offers a variety of API objects.

The following list describes some of the commonly used API objects:

ULDatabaseManager

Provides methods for managing databases and connections.

ULConnection

Represents a connection to an UltraLite database. You can create one or more ULConnection objects.

ULTable

Provides direct access to tables in the database.

ULPreparedStatement, ULResultSet, and ULResultSetSchema

Create Dynamic SQL statements, make queries, execute INSERT, UPDATE, and DELETE statements, and attain programmatic control over database result sets.

Header File

- `ulcpp.h`

In this section:

Represents a table in an UltraLite database.

ULTableSchema Class [page 168]
Represents the schema of an UltraLite table.

ul_column_default_type Enumeration [page 178]
Identifies a column default type.

ul_column_name_type Enumeration [page 179]
Specifies values that control how a column name is retrieved when describing a result set.

ul_index_flag Enumeration [page 180]
Flags (bit fields) which identify properties of an index.

ul_table_sync_type Enumeration [page 181]
Identifies a table synchronization type.

UL_BLOB_CONTINUE Variable [page 182]
Used when reading data with the ULResultSet.GetStringChunk or ULResultSet.GetByteChunk methods.

ul_index_iter_start Variable [page 183]
Used by the GetNextIndex method to initialize index iteration in a table.

ul_publication_iter_start Variable [page 183]
Used by the GetNextPublication method to initialize publication iteration in a database.

ul_table_iter_start Variable [page 183]
Used by the GetNextTable method to initialize table iteration in a database.

1.1 ULConnection Class

Represents a connection to an UltraLite database.

Syntax

```
public class ULConnection
```

Members

All members of ULConnection, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual ul_u_long	CancelGetNotification(const char *) [page 14]	Cancels any pending get-notification calls on all queues matching the given name.

Modifier and Type	Method	Description
public virtual bool	ChangeEncryptionKey(const char *) [page 15]	Changes the database encryption key for an UltraLite database.
public virtual bool	Checkpoint() [page 15]	Performs a checkpoint operation, flushing any pending committed transactions to the database.
public virtual void	Close(ULError *) [page 16]	Destroys this connection and any remaining associated objects.
public virtual bool	Commit() [page 16]	Commits the current transaction.
public virtual ul_u_long	CountUploadRows(const char *, ul_u_long) [page 17]	Counts the number of rows that need to be uploaded for synchronization.
public virtual bool	CreateNotificationQueue(const char *, const char *) [page 18]	Creates an event notification queue for this connection.
public virtual bool	DeclareEvent(const char *) [page 19]	Declares an event which can then be registered for and triggered.
public virtual bool	DestroyNotificationQueue(const char *) [page 19]	Destroys the given event notification queue.
public virtual bool	ExecuteScalar(void *, size_t, ul_column_storage_type, const char *, ...) [page 20]	Executes a SQL SELECT statement directly, returning a single result.
public virtual bool	ExecuteScalarV(void *, size_t, ul_column_storage_type, const char *, va_list) [page 22]	Executes a SQL SELECT statement string, along with a list of substitution values.
public virtual bool	ExecuteStatement(const char *) [page 23]	Executes a SQL statement string directly.
public virtual ul_u_long	GetChildObjectCount() [page 24]	Gets the number of currently open child objects on the connection.
public virtual const char *	GetDatabaseProperty(const char *) [page 25]	Obtains the value of a database property.
public virtual ul_u_long	GetDatabasePropertyInt(const char *) [page 25]	Obtains the integer value of a database property.
public virtual ULDatabaseSchema *	GetDatabaseSchema() [page 26]	Returns an object pointer used to query the schema of the database.
public virtual bool	GetLastDownloadTime(const char *, DECL_DATETIME *) [page 26]	Obtains the last time a specified publication was downloaded.
public virtual const ULError *	GetLastError() [page 27]	Returns the error information associated with the last call.
public virtual ul_u_big	GetLastIdentity() [page 28]	Gets the @@identity value.
public virtual const char *	GetNotification(const char *, ul_u_long) [page 28]	Reads an event notification.
public virtual const char *	GetNotificationParameter(const char *, const char *) [page 29]	Gets a parameter for the event notification just read by the GetNotification method.

Modifier and Type	Method	Description
public virtual SQLCA *	GetSqlca() [page 30]	Gets the communication area associated with this connection.
public virtual bool	GetSyncResult(ul_sync_result *) [page 30]	Gets the result of the last synchronization.
public virtual void *	GetUserPointer() [page 31]	Gets the pointer value last set by the SetUserPointer method.
public virtual ul_u_short	GlobalAutoincUsage() [page 31]	Obtains the percent of the default values used in all the columns that have global autoincrement defaults.
public virtual bool	GrantConnectTo(const char *, const char *) [page 32]	Grants access to an UltraLite database for a new or existing user ID with the given password.
public virtual void	InitSyncInfo(ul_sync_info *) [page 33]	Initializes the synchronization information structure.
public virtual ULTable *	OpenTable(const char *, const char *) [page 33]	Opens a table.
public virtual ULPreparedStatement *	PrepareStatement(const char *) [page 34]	Prepares a SQL statement.
public virtual bool	RegisterForEvent(const char *, const char *, const char *, bool) [page 34]	Registers or unregisters a queue to receive notifications of an event.
public virtual bool	ResetLastDownloadTime(const char *) [page 35]	Resets the last download time of a publication so that the application resynchronizes previously downloaded data.
public virtual bool	RevokeConnectFrom(const char *) [page 36]	Revokes access from an UltraLite database for a user ID.
public virtual bool	Rollback() [page 37]	Rolls back the current transaction.
public virtual bool	RollbackPartialDownload() [page 37]	Rolls back the changes from a failed synchronization.
public virtual ul_u_long	SendNotification(const char *, const char *, const char *) [page 38]	Sends a notification to all queues matching the given name.
public virtual bool	SetDatabaseOption(const char *, const char *) [page 39]	Sets the specified database option.
public virtual bool	SetDatabaseOptionInt(const char *, ul_u_long) [page 39]	Sets a database option.
public virtual void	SetSynchronizationCallback(ul_sync_observer_fn, void *) [page 40]	Sets the callback to be invoked while performing a synchronization.
public virtual bool	SetSyncInfo(char const *, ul_sync_info *) [page 40]	Creates a synchronization profile using the given name based on the given ul_sync_info structure.
public virtual void *	SetUserPointer(void *) [page 41]	Sets an arbitrary pointer value in the connection for use by the calling application.

Modifier and Type	Method	Description
public virtual bool	StartSynchronizationDelete() [page 41]	Sets START SYNCHRONIZATION DELETE for this connection.
public virtual bool	StopSynchronizationDelete() [page 42]	Sets STOP SYNCHRONIZATION DELETE for this connection.
public virtual bool	Synchronize(ul_sync_info *) [page 42]	Initiates synchronization in an UltraLite application.
public virtual bool	SynchronizeFromProfile(const char *, const char *, ul_sync_observer_fn, void *) [page 43]	Synchronizes the database using the given profile and merge parameters.
public virtual ul_u_long	TriggerEvent(const char *, const char *) [page 44]	Triggers a user-defined event and sends notifications to all registered queues.
public virtual bool	ValidateDatabase(ul_u_short, ul_validate_callback_fn, void *, const char *) [page 45]	Validates the database on this connection.

In this section:

CancelGetNotification(const char *) Method [page 14]
Cancels any pending get-notification calls on all queues matching the given name.

ChangeEncryptionKey(const char *) Method [page 15]
Changes the database encryption key for an UltraLite database.

Checkpoint() Method [page 15]
Performs a checkpoint operation, flushing any pending committed transactions to the database.

Close(ULError *) Method [page 16]
Destroys this connection and any remaining associated objects.

Commit() Method [page 16]
Commits the current transaction.

CountUploadRows(const char *, ul_u_long) Method [page 17]
Counts the number of rows that need to be uploaded for synchronization.

CreateNotificationQueue(const char *, const char *) Method [page 18]
Creates an event notification queue for this connection.

DeclareEvent(const char *) Method [page 19]
Declares an event which can then be registered for and triggered.

DestroyNotificationQueue(const char *) Method [page 19]
Destroys the given event notification queue.

ExecuteScalar(void *, size_t, ul_column_storage_type, const char *, ...) Method [page 20]
Executes a SQL SELECT statement directly, returning a single result.

ExecuteScalarV(void *, size_t, ul_column_storage_type, const char *, va_list) Method [page 22]
Executes a SQL SELECT statement string, along with a list of substitution values.

ExecuteStatement(const char *) Method [page 23]
Executes a SQL statement directly.

GetChildObjectCount() Method [page 24]

Rolls back the current transaction.

RollbackPartialDownload() Method [page 37]
Rolls back the changes from a failed synchronization.

SendNotification(const char *, const char *, const char *) Method [page 38]
Sends a notification to all queues matching the given name.

SetDatabaseOption(const char *, const char *) Method [page 39]
Sets the specified database option.

SetDatabaseOptionInt(const char *, ul_u_long) Method [page 39]
Sets a database option.

SetSynchronizationCallback(ul_sync_observer_fn, void *) Method [page 40]
Sets the callback to be invoked while performing a synchronization.

SetSyncInfo(char const *, ul_sync_info *) Method [page 40]
Creates a synchronization profile using the given name based on the given ul_sync_info structure.

SetUserPointer(void *) Method [page 41]
Sets an arbitrary pointer value in the connection for use by the calling application.

StartSynchronizationDelete() Method [page 41]
Sets START SYNCHRONIZATION DELETE for this connection.

StopSynchronizationDelete() Method [page 42]
Sets STOP SYNCHRONIZATION DELETE for this connection.

Synchronize(ul_sync_info *) Method [page 42]
Initiates synchronization in an UltraLite application.

SynchronizeFromProfile(const char *, const char *, ul_sync_observer_fn, void *) Method [page 43]
Synchronizes the database using the given profile and merge parameters.

TriggerEvent(const char *, const char *) Method [page 44]
Triggers a user-defined event and sends notifications to all registered queues.

ValidateDatabase(ul_u_short, ul_validate_callback_fn, void *, const char *) Method [page 45]
Validates the database on this connection.

1.1.1 CancelGetNotification(const char *) Method

Cancels any pending get-notification calls on all queues matching the given name.

Syntax

```
public virtual ul_u_long CancelGetNotification (const char * queueName)
```

Parameters

queueName The name of the queue.

Returns

The number of affected queues. (not the number of blocked reads necessarily)

1.1.2 ChangeEncryptionKey(const char *) Method

Changes the database encryption key for an UltraLite database.

Syntax

```
public virtual bool ChangeEncryptionKey (const char * newKey)
```

Parameters

newKey The new encryption key for the database.

Returns

True on success; otherwise, returns false.

Remarks

Applications that call this method must first ensure that the user has either synchronized the database or created a reliable backup copy of the database. It is important to have a reliable backup of the database because the ChangeEncryptionKey method is an operation that must run to completion. When the database encryption key is changed, every row in the database is first decrypted with the old key and then encrypted with the new key and rewritten. This operation is not recoverable. If the encryption change operation does not complete, the database is left in an invalid state and you cannot access it again.

1.1.3 Checkpoint() Method

Performs a checkpoint operation, flushing any pending committed transactions to the database.

Syntax

```
public virtual bool Checkpoint ()
```

Returns

True on success; otherwise, returns false.

Remarks

Any current transaction is not committed by calling the Checkpoint method. This method is used in conjunction with deferring automatic transaction checkpoints (using the *commit_flush* connection parameter) as a performance enhancement.

The Checkpoint method ensures that all pending committed transactions have been written to the database.

1.1.4 Close(ULError *) Method

Destroys this connection and any remaining associated objects.

Syntax

```
public virtual void Close (ULError * error)
```

Parameters

error An optional ULError object to receive error information.

1.1.5 Commit() Method

Commits the current transaction.

Syntax

```
public virtual bool Commit ()
```

Returns

True on success; otherwise, returns false.

1.1.6 CountUploadRows(const char *, ul_u_long) Method

Counts the number of rows that need to be uploaded for synchronization.

Syntax

```
public virtual ul_u_long CountUploadRows (
    const char * pubList,
    ul_u_long threshold
)
```

Parameters

pubList A string containing a comma-separated list of publications to check. An empty string (the UL_SYNC_ALL macro) implies all tables except tables marked as "no sync". A string containing just an asterisk (the UL_SYNC_ALL_PUBS macro) implies all tables referred to in any publication. Some tables may not be part of any publication and are not included if this value is "*".

threshold Determines the maximum number of rows to count, thereby limiting the amount of time taken by the call. A threshold of 0 corresponds to no limit (that is, count all rows that need to be synchronized) and a threshold of 1 can be used to quickly determine if any rows need to be synchronized.

Returns

The number of rows that need to be synchronized, either in a specified set of publications or in the whole database.

Remarks

Use this method to prompt users to synchronize, or determine when automatic background synchronization should take place.

The following call checks the entire database for the total number of rows to be synchronized:

```
count = conn->CountUploadRows( UL_SYNC_ALL, 0 );
```

The following call checks publications PUB1 and PUB2 for a maximum of 1000 rows:

```
count = conn->CountUploadRows( "PUB1,PUB2", 1000 );
```

The following call checks to see if any rows need to be synchronized in publications PUB1 and PUB2:

```
anyToSync = conn->CountUploadRows( "PUB1,PUB2", 1 ) != 0;
```

1.1.7 CreateNotificationQueue(const char *, const char *) Method

Creates an event notification queue for this connection.

Syntax

```
public virtual bool CreateNotificationQueue (
    const char * name,
    const char * parameters
)
```

Parameters

name The name for the new queue.

parameters Reserved. Set to NULL.

Returns

True on success; otherwise, returns false.

Remarks

Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive (a separate instance of) the notification. Names are case insensitive. A default queue is created on demand for each connection when calling the RegisterForEvent method if no queue is specified. This call fails with an error if the name already exists or isn't valid.

Related Information

RegisterForEvent(const char *, const char *, const char *, bool) Method [page 34]

1.1.8 DeclareEvent(const char *) Method

Declares an event which can then be registered for and triggered.

Syntax

```
public virtual bool DeclareEvent (const char * eventName)
```

Parameters

eventName The name for the new user-defined event.

Returns

True if the event was declared successfully; otherwise, returns false if the name is already used or not valid.

Remarks

UltraLite predefines some system events triggered by operations on the database or the environment. This method declares user-defined events. User-defined events are triggered with the TriggerEvent method. The event name must be unique. Names are case insensitive.

Related Information

TriggerEvent(const char *, const char *) Method [page 44]

1.1.9 DestroyNotificationQueue(const char *) Method

Destroys the given event notification queue.

Syntax

```
public virtual bool DestroyNotificationQueue (const char * name)
```

Parameters

name The name of the queue to destroy.

Returns

True on success; otherwise, returns false.

Remarks

A warning is signaled if unread notifications remain in the queue. Unread notifications are discarded. A connection's default event queue, if created, is destroyed when the connection is closed.

1.1.10 ExecuteScalar(void *, size_t, ul_column_storage_type, const char *, ...) Method

Executes a SQL SELECT statement directly, returning a single result.

Syntax

```
public virtual bool ExecuteScalar (
    void * dstPtr,
    size_t dstSize,
    ul_column_storage_type dstType,
    const char * sql,
    ...
)
```

Parameters

dstPtr A pointer to a variable of the required type to receive the value.

dstSize The size of variable to receive value, if applicable.

dstType The type of value to retrieve. This value must match the variable type.

sql The SELECT statement, optionally containing '?' parameters. For each '?' substitution parameter, supply a corresponding string (char *) parameter.

Returns

True if the query is successfully executed and a value is successfully retrieved; otherwise, returns false when a value is not fetched. Check the SQLCODE error code to determine why false is returned. The selected value is NULL if no warning or error (SQLE_NOERROR) is indicated.

Remarks

The dstPtr value must point to a variable of the correct type, matching the dstType value. The dstSize parameter is only required for variable-sized values, such as strings and binaries, and is otherwise ignored. The variable list of parameter values must correspond to parameters in the statement, and all values are assumed to be strings. (internally, UltraLite casts the parameter values as required for the statement)

The following types are supported:

UL_TYPE_BIT/UL_TYPE_TINY

Use variable type ul_byte (8 bit, unsigned).

UL_TYPE_U_SHORT/UL_TYPE_S_SHORT

Use variable type ul_u_short/ul_s_short (16 bit).

UL_TYPE_U_LONG/UL_TYPE_S_LONG

Use variable type ul_u_long/ul_s_long (32 bit).

UL_TYPE_U_BIG/UL_TYPE_S_BIG

Use variable type ul_u_big/ul_s_big (64 bit).

UL_TYPE_DOUBLE

Use variable type ul_double (double).

UL_TYPE_REAL

Use variable type ul_real (float).

UL_TYPE_BINARY

Use variable type ul_binary and specify *dstSize* (as in GetBinary()).

UL_TYPE_TIMESTAMP_STRUCT

Use variable type DECL_DATETIME.

UL_TYPE_CHAR

Use variable type char [] (a character buffer), and set *dstSize* to the size of the buffer (as in GetString()).

UL_TYPE_WCHAR

Use variable type ul_wchar [] (a wide character buffer), and set *dstSize* to the size of the buffer (as in GetString()).

UL_TYPE_TCHAR

Same as UL_TYPE_CHAR or UL_TYPE_WCHAR, depending on which version of the method is called.

The following example demonstrates integer fetching:

```
ul_u_long       val;
```

```
ok = conn->ExecuteScalar( &val, 0, UL_TYPE_U_LONG,
   "SELECT count(*) FROM t WHERE col LIKE ?", "ABC%" );
```

The following example demonstrates string fetching:

```
char    val[40];
ok = conn->ExecuteScalar( &val, sizeof(val), UL_TYPE_CHAR,
   "SELECT uuidtostr( newid() )" );
```

1.1.11 ExecuteScalarV(void *, size_t, ul_column_storage_type, const char *, va_list) Method

Executes a SQL SELECT statement string, along with a list of substitution values.

Syntax

```
public virtual bool ExecuteScalarV (
    void * dstPtr,
    size_t dstSize,
    ul_column_storage_type dstType,
    const char * sql,
    va_list args
)
```

Parameters

dstPtr A pointer to a variable of the required type to receive the value.
dstSize The size of variable to receive value, if applicable.
dstType The type of value to retrieve. This value must match the variable type.
sql The SELECT statement, optionally containing '?' parameters.
args A list of string (char *) values to substitute.

Returns

True if the query is successfully executed and a value is successfully retrieved; otherwise, returns false when a value is not fetched. Check the SQLCODE error code to determine why false is returned. The selected value is NULL if no warning or error (SQLE_NOERROR) is indicated.

Remarks

The dstPtr value must point to a variable of the correct type, matching the dstType value. The dstSize parameter is only required for variable-sized values, such as strings and binaries, and is otherwise ignored. The

variable list of parameter values must correspond to parameters in the statement, and all values are assumed to be strings. (internally, UltraLite casts the parameter values as required for the statement)

The following types are supported:

UL_TYPE_BIT/UL_TYPE_TINY

Use variable type ul_byte (8 bit, unsigned).

UL_TYPE_U_SHORT/UL_TYPE_S_SHORT

Use variable type ul_u_short/ul_s_short (16 bit).

UL_TYPE_U_LONG/UL_TYPE_S_LONG

Use variable type ul_u_long/ul_s_long (32 bit).

UL_TYPE_U_BIG/UL_TYPE_S_BIG

Use variable type ul_u_big/ul_s_big (64 bit).

UL_TYPE_DOUBLE

Use variable type ul_double (double).

UL_TYPE_REAL

Use variable type ul_real (float).

UL_TYPE_BINARY

Use variable type ul_binary and specify *dstSize* (as in GetBinary()).

UL_TYPE_TIMESTAMP_STRUCT

Use variable type DECL_DATETIME.

UL_TYPE_CHAR

Use variable type char [] (a character buffer), and set *dstSize* to the size of the buffer (as in GetString()).

UL_TYPE_WCHAR

Use variable type ul_wchar [] (a wide character buffer), and set *dstSize* to the size of the buffer (as in GetString()).

UL_TYPE_TCHAR

Same as UL_TYPE_CHAR or UL_TYPE_WCHAR, depending on which version of the method is called.

1.1.12 ExecuteStatement(const char *) Method

Executes a SQL statement directly.

Syntax

```
public virtual bool ExecuteStatement (const char * sql)
```

Parameters

sql The SQL statement to execute.

Returns

True on success; otherwise, returns false.

Remarks

Use this method to execute a non-SELECT statement directly.

Use the PrepareStatement method to execute a statement repeatedly with variable parameters, or to fetch multiple results.

Related Information

PrepareStatement(const char *) Method [page 34]

1.1.13 GetChildObjectCount() Method

Gets the number of currently open child objects on the connection.

Syntax

```
public virtual ul_u_long GetChildObjectCount ()
```

Returns

The number of currently open child objects.

Remarks

This method can be used to detect object leaks.

1.1.14 GetDatabaseProperty(const char *) Method

Obtains the value of a database property.

Syntax

```
public virtual const char * GetDatabaseProperty (const char * propName)
```

Parameters

propName The name of the property being requested.

Returns

A pointer to a string buffer containing the database property value is returned when run successfully; otherwise, returns NULL.

Remarks

The returned value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you must make a copy of the value if you need to save it.

Example

The following example illustrates how to get the value of the CharSet database property.

```
const char * charset = GetDatabaseProperty( "CharSet" );
```

1.1.15 GetDatabasePropertyInt(const char *) Method

Obtains the integer value of a database property.

Syntax

```
public virtual ul_u_long GetDatabasePropertyInt (const char * propName)
```

Parameters

propName The name of the property being requested.

Returns

If successful, the integer value of the property; otherwise, returns 0.

Example

The following example illustrates how to get the value of the ConnCount database property.

```
unsigned connectionCount = GetDatabasePropertyInt( "ConnCount" );
```

1.1.16 GetDatabaseSchema() Method

Returns an object pointer used to query the schema of the database.

Syntax

```
public virtual ULDatabaseSchema * GetDatabaseSchema ()
```

Returns

A ULDatabaseSchema object used to query the schema of the database.

1.1.17 GetLastDownloadTime(const char *, DECL_DATETIME *) Method

Obtains the last time a specified publication was downloaded.

Syntax

```
public virtual bool GetLastDownloadTime (
    const char * publication,
    DECL_DATETIME * value
)
```

Parameters

publication The publication name.
value A pointer to the DECL_DATETIME structure to be populated. The value of January 1, 1900 indicates that the publication has yet to be synchronized, or the time was reset.

Returns

True when the value is successfully populated by the last download time of the publication specified; otherwise, returns false.

Remarks

The following call populates the dt structure with the date and time that the 'pub1' publication was downloaded:

```
DECL_DATETIME dt;
ok = conn->GetLastDownloadTime( "pub1", &dt );
```

1.1.18 GetLastError() Method

Returns the error information associated with the last call.

Syntax

```
public virtual const ULError * GetLastError ()
```

Returns

A pointer to the ULError object with information associated with the last call.

Remarks

The error object whose address is returned remains valid while the connection is open, but not updated automatically on subsequent calls. You must call GetLastError to retrieve updated status information.

Related Information

ULError Class [page 64]

1.1.19 GetLastIdentity() Method

Gets the @@identity value.

Syntax

```
public virtual ul_u_big GetLastIdentity ()
```

Returns

The last value inserted into an autoincrement or global autoincrement column

Remarks

This value is the last value inserted into an autoincrement or global autoincrement column for the database. This value is not recorded when the database is shutdown, so calling this method before any autoincrement values have been inserted returns 0.

Note

The last value inserted may have been on another connection.

1.1.20 GetNotification(const char *, ul_u_long) Method

Reads an event notification.

Syntax

```
public virtual const char * GetNotification (
    const char * queueName,
    ul_u_long waitms
)
```

Parameters

queueName The queue to read or NULL for the default connection queue.
waitms The time, in milliseconds to wait (block) before returning.

Returns

The name of the event read or NULL on error.

Remarks

This call blocks until a notification is received or until the given wait period expires. To wait indefinitely, set the waitms parameter to UL_READ_WAIT_INFINITE. To cancel a wait, send another notification to the given queue or use the CancelGetNotification method. Use the GetNotificationParameter method after reading a notification to retrieve additional parameters by name.

Related Information

CancelGetNotification(const char *) Method [page 14]
GetNotificationParameter(const char *, const char *) Method [page 29]

1.1.21 GetNotificationParameter(const char *, const char *) Method

Gets a parameter for the event notification just read by the GetNotification method.

Syntax

```
public virtual const char * GetNotificationParameter (
    const char * queueName,
    const char * parameterName
)
```

Parameters

queueName The queue to read or NULL for default connection queue.
parameterName The name of the parameter to read (or "*").

Returns

The parameter value or NULL on error.

Remarks

Only the parameters from the most recently read notification on the given queue are available. Parameters are retrieved by name. A parameter name of "*" retrieves the entire parameter string.

Related Information

GetNotification(const char *, ul_u_long) Method [page 28]

1.1.22 GetSqlca() Method

Gets the communication area associated with this connection.

Syntax

```
public virtual SQLCA * GetSqlca ()
```

Returns

A pointer to the SQLCA object for this connection.

1.1.23 GetSyncResult(ul_sync_result *) Method

Gets the result of the last synchronization.

Syntax

```
public virtual bool GetSyncResult (ul_sync_result * syncResult)
```

Parameters

syncResult A pointer to the ul_sync_result structure to be populated.

Returns

True on success, otherwise false.

1.1.24 GetUserPointer() Method

Gets the pointer value last set by the SetUserPointer method.

Syntax

```
public virtual void * GetUserPointer ()
```

Related Information

SetUserPointer(void *) Method [page 41]

1.1.25 GlobalAutoincUsage() Method

Obtains the percent of the default values used in all the columns that have global autoincrement defaults.

Syntax

```
public virtual ul_u_short GlobalAutoincUsage ()
```

Returns

The percent of the global autoincrement values used by the counter.

Remarks

If the database contains more than one column with this default, this value is calculated for all columns and the maximum is returned. For example, a return value of 99 indicates that very few default values remain for at least one of the columns.

1.1.26 GrantConnectTo(const char *, const char *) Method

Grants access to an UltraLite database for a new or existing user ID with the given password.

Syntax

```
public virtual bool GrantConnectTo (
    const char * uid,
    const char * pwd
)
```

Parameters

uid A character array that holds the user ID. The maximum length is 31 characters.
pwd A character array that holds the password for the user ID.

Returns

True on success; otherwise, returns false.

Remarks

This method updates the password for an existing user when you specify an existing user ID.

Related Information

RevokeConnectFrom(const char *) Method [page 36]

1.1.27 InitSyncInfo(ul_sync_info *) Method

Initializes the synchronization information structure.

Syntax

```
public virtual void InitSyncInfo (ul_sync_info * info)
```

Parameters

info A pointer to the ul_sync_info structure that holds the synchronization parameters.

Remarks

Call this method before setting the values of fields in the ul_sync_info structure.

1.1.28 OpenTable(const char *, const char *) Method

Opens a table.

Syntax

```
public virtual ULTable * OpenTable (
    const char * tableName,
    const char * indexName
)
```

Parameters

tableName The name of the table to open.
indexName The name of the index to open the table on. Pass NULL to open on the primary key and the empty string to open the table unordered.

Returns

The ULTable object when the call is successful; otherwise, returns NULL.

Remarks

The cursor position is set before the first row when the application first opens a table.

1.1.29 PrepareStatement(const char *) Method

Prepares a SQL statement.

Syntax

```
public virtual ULPreparedStatement * PrepareStatement (const char * sql)
```

Parameters

sql The SQL statement to prepare.

Returns

The ULPreparedStatement object on success; otherwise, returns NULL.

1.1.30 RegisterForEvent(const char *, const char *, const char *, bool) Method

Registers or unregisters a queue to receive notifications of an event.

Syntax

```
public virtual bool RegisterForEvent (
    const char * eventName,
    const char * objectName,
    const char * queueName,
    bool register_not_unreg
)
```

Parameters

eventName The system- or user-defined event to register for.

objectName The object to which the event applies. (for example, a table name).
queueName NULL means use the default connection queue.
register_not_unreg Set true to register, or false to unregister.

Returns

True if the registration succeeded; otherwise, returns false if the queue or event does not exist.

Remarks

If no queue name is supplied, the default connection queue is implied, and created if required. Certain system events allow you to specify an object name to which the event applies. For example, the TableModified event can specify the table name. Unlike the SendNotification method, only the specific queue registered receives notifications of the event. Other queues with the same name on different connections do not receive notifications, unless they are also explicitly registered.

The predefined system events are:

TableModified

Triggered when rows in a table are inserted, updated, or deleted. One notification is sent per request, no matter how many rows were affected by the request. The object_name parameter specifies the table to monitor. A value of "*" means all tables in the database. This event has a parameter named table_name whose value is the name of the modified table.

Commit

Triggered after any commit completes. This event has no parameters.

SyncComplete

Triggered after synchronization completes. This event has no parameters.

1.1.31 ResetLastDownloadTime(const char *) Method

Resets the last download time of a publication so that the application resynchronizes previously downloaded data.

Syntax

```
public virtual bool ResetLastDownloadTime (const char * pubList)
```

Parameters

pubList A string containing a comma-separated list of publications to reset. An empty string means all tables except tables marked as "no sync". A string containing just an asterisk ("*") denotes all publications. Some tables may not be part of any publication and are not included if this value is "*".

Returns

True on success; otherwise, returns false.

Remarks

The following method call resets the last download time for all tables:

```
conn->ResetLastDownloadTime( "" );
```

1.1.32 RevokeConnectFrom(const char *) Method

Revokes access from an UltraLite database for a user ID.

Syntax

```
public virtual bool RevokeConnectFrom (const char * uid)
```

Parameters

uid A character array holding the user ID to be excluded from database access.

Returns

True on success, otherwise false.

1.1.33 Rollback() Method

Rolls back the current transaction.

Syntax

```
public virtual bool Rollback ()
```

Returns

True on success, otherwise false.

1.1.34 RollbackPartialDownload() Method

Rolls back the changes from a failed synchronization.

Syntax

```
public virtual bool RollbackPartialDownload ()
```

Returns

True on success, otherwise false.

Remarks

When using resumable downloads (synchronizing with the keep-partial-download option turned on), and a communication error occurs during the download phase of synchronization, UltraLite retains the changes which were downloaded (so the synchronization can resume from the place it was interrupted). Use this method to discard this partial download when you no longer wish to attempt resuming.

This method has effect only when using resumable downloads.

1.1.35 SendNotification(const char *, const char *, const char *) Method

Sends a notification to all queues matching the given name.

Syntax

```
public virtual ul_u_long SendNotification (
    const char * queueName,
    const char * eventName,
    const char * parameters
)
```

Parameters

queueName The target queue name (or "*").
eventName The identity for notification.
parameters Optional parameters option list.

Returns

The number of notifications sent. (the number of matching queues)

Remarks

This includes any such queue on the current connection. This call does not block. Use the special queue name "*" to send to all queues. The given event name does not need to correspond to any system or user- defined event; it is simply passed through to identify the notification when read and has meaning only to the sender and receiver.

The *parameters* value specifies a semicolon delimited name=value pairs option list. After the notification is read, the parameter values are read with the GetNotificationParameter method.

Related Information

GetNotificationParameter(const char *, const char *) Method [page 29]

1.1.36 SetDatabaseOption(const char *, const char *) Method

Sets the specified database option.

Syntax

```
public virtual bool SetDatabaseOption (
    const char * optName,
    const char * value
)
```

Parameters

optName The name of the option being set.
value The new value of the option.

Returns

True on success, otherwise false.

1.1.37 SetDatabaseOptionInt(const char *, ul_u_long) Method

Sets a database option.

Syntax

```
public virtual bool SetDatabaseOptionInt (
    const char * optName,
    ul_u_long value
)
```

Parameters

optName The name of the option being set.
value The new value of the option.

Returns

True on success; otherwise, returns false.

1.1.38 SetSynchronizationCallback(ul_sync_observer_fn, void *) Method

Sets the callback to be invoked while performing a synchronization.

Syntax

```
public virtual void SetSynchronizationCallback (
    ul_sync_observer_fn callback,
    void * userData
)
```

Parameters

callback The ul_sync_observer_fn callback.
userData User context information passed to the callback.

1.1.39 SetSyncInfo(char const *, ul_sync_info *) Method

Creates a synchronization profile using the given name based on the given ul_sync_info structure.

Syntax

```
public virtual bool SetSyncInfo (
    char const * profileName,
    ul_sync_info * info
)
```

Parameters

profileName The name of the synchronization profile.
info A pointer to the ul_sync_info structure that holds the synchronization parameters.

Returns

True on success; otherwise, returns false.

Remarks

The synchronization profile replaces any previous profile with the same name. The named profile is deleted by specifying a null pointer for the structure.

1.1.40 SetUserPointer(void *) Method

Sets an arbitrary pointer value in the connection for use by the calling application.

Syntax

```
public virtual void * SetUserPointer (void * ptr)
```

Returns

The previously set pointer value.

Remarks

This can be used to associate application data with the connection.

1.1.41 StartSynchronizationDelete() Method

Sets START SYNCHRONIZATION DELETE for this connection.

Syntax

```
public virtual bool StartSynchronizationDelete ()
```

Returns

True on success, otherwise false.

1.1.42 StopSynchronizationDelete() Method

Sets STOP SYNCHRONIZATION DELETE for this connection.

Syntax

```
public virtual bool StopSynchronizationDelete ()
```

Returns

True on success, otherwise false.

1.1.43 Synchronize(ul_sync_info *) Method

Initiates synchronization in an UltraLite application.

Syntax

```
public virtual bool Synchronize (ul_sync_info * info)
```

Parameters

info A pointer to the ul_sync_info structure that holds the synchronization parameters.

Returns

True on success; otherwise, returns false.

Remarks

This method initiates synchronization with the MobiLink server. This method does not return until synchronization is complete, however additional threads on separate connections may continue to access the database during synchronization.

Before calling this method, enable the protocol and encryption you are using with methods in the ULDatabaseManager class. For example, when using "HTTP", call the ULDatabaseManager.EnableHttpSynchronization method.

The following example demonstrates database synchronization:

```
ul_sync_info info;
conn->InitSyncInfo( &info );
info.user_name = "my_user";
info.version = "myapp_1_2";
info.stream = "HTTP";
info.stream_parms = "host=myserver.com";
conn->Synchronize( &info );
```

Related Information

EnableHttpSynchronization() Method [page 52]

1.1.44 SynchronizeFromProfile(const char *, const char *, ul_sync_observer_fn, void *) Method

Synchronizes the database using the given profile and merge parameters.

Syntax

```
public virtual bool SynchronizeFromProfile (
    const char * profileName,
    const char * mergeParms,
    ul_sync_observer_fn observer,
    void * userData
)
```

Parameters

profileName The name of the profile to synchronize.
mergeParms Merge parameters for the synchronization.
observer The observer callback to send status updates to.
userData User context data passed to callback.

Returns

True on success; otherwise, returns false.

Remarks

This method is identical to executing the SYNCHRONIZE statement.

Related Information

Synchronize(ul_sync_info *) Method [page 42]

1.1.45 TriggerEvent(const char *, const char *) Method

Triggers a user-defined event and sends notifications to all registered queues.

Syntax

```
public virtual ul_u_long TriggerEvent (
    const char * eventName,
    const char * parameters
)
```

Parameters

eventName The name of the system or user-defined event to trigger.
parameters Optional parameters option list.

Returns

The number of event notifications sent.

Remarks

The *parameters* value specifies a semicolon delimited name=value pairs option list. After the notification is read, the parameter values are read with GetNotificationParameter().

Related Information

GetNotificationParameter(const char *, const char *) Method [page 29]

1.1.46 ValidateDatabase(ul_u_short, ul_validate_callback_fn, void *, const char *) Method

Validates the database on this connection.

Syntax

```
public virtual bool ValidateDatabase (
    ul_u_short flags,
    ul_validate_callback_fn fn,
    void * user_data,
    const char * tableName
)
```

Parameters

flags Flags controlling the type of validation. See the example below.
fn Function to receive validation progress information.
user_data User data to send back to the caller via the callback.
tableName Optional. A specific table to validate.

Returns

True on success; otherwise, returns false.

Remarks

Tables, indexes, and database pages can be validated depending on the flags passed to this routine. To receive information during the validation, implement a callback function and pass the address to this routine. To limit the validation to a specific table, pass in the table name or ID as the last parameter.

The flags parameter is combination of the following values:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Example

The following example demonstrates table and index validation in express mode:

```
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

1.2 ULDatabaseManager Class

Manages connections and databases.

Syntax

```
public class ULDatabaseManager
```

Members

All members of ULDatabaseManager, including inherited members.

Methods

Modifier and Type	Method	Description
public static ULConnection *	CreateDatabase(const char *, const char *, ULError *) [page 49]	Creates a new database.
public static bool	DropDatabase(const char *, ULError *) [page 50]	Erases an existing database that is not currently running.
public static void	EnableAesDBEncryption() [page 50]	Enables AES database encryption.

Modifier and Type	Method	Description
public static void	EnableAesFipsDBEncryption() [page 51]	Enables FIPS 140-2 certified AES database encryption.
public static void	EnableAllSynchronization() [page 51]	Enables all four types of synchronization: TCPIP, HTTP, TLS, and HTTPS.
public static void	EnableHttpsSynchronization() [page 52]	Enables HTTPS synchronization.
public static void	EnableHttpSynchronization() [page 52]	Enables HTTP synchronization.
public static void	EnableRsaE2ee() [page 52]	Enables RSA end-to-end encryption.
public static void	EnableRsaFipsE2ee() [page 53]	Enables FIPS 140-2 certified RSA end-to-end encryption.
public static void	EnableRsaFipsSyncEncryption() [page 53]	Enables FIPS 140-2 certified RSA synchronization encryption for SSL or TLS streams.
public static void	EnableRsaSyncEncryption() [page 54]	Enables RSA synchronization encryption.
public static void	EnableTcpipSynchronization() [page 54]	Enables TCP/IP synchronization.
public static void	EnableTlsSynchronization() [page 55]	Enables TLS synchronization.
public static void	EnableZlibSyncCompression() [page 55]	Enables Zlib compression for a synchronization stream.
public static void	Fini() [page 55]	Finalizes the UltraLite runtime.
public static bool	Init() [page 56]	Initializes the UltraLite runtime.
public static ULConnection *	OpenConnection(const char *, ULError *, void *) [page 56]	Opens a new connection to an existing database.
public static void	SetErrorCallback(ul_cpp_error_callback_fn, void *) [page 57]	Sets the callback to be invoked when an error occurs.
public static bool	ValidateDatabase(const char *, ul_u_short, ul_validate_callback_fn, void *, ULError *) [page 58]	Performs low level and index validation on a database.

Remarks

The Init method must be called in a thread-safe environment before any other calls can be made. The Fini method must be called in a similarly thread-safe environment when finished.

> **i** Note
>
> This class is static. Do not create an instance of it.

In this section:

1.2.1 CreateDatabase(const char *, const char *, ULError *) Method

Creates a new database.

Syntax

```
public static ULConnection * CreateDatabase (
    const char * connParms,
    const char * createParms,
    ULError * error
)
```

Parameters

connParms A semicolon separated string of connection parameters, which are set as keyword=value pairs. The connection string must include the name of the database. These parameters are the same set of parameters that can be specified when you connect to a database.

createParms A semicolon separated string of database creation parameters, which are set as keyword value pairs. For example: page_size=2048;obfuscate=yes.

error An optional ULError object to receive error information.

Returns

A ULConnection object to the new database is returned if the database was created successfully. NULL is returned if the method fails. Failure is usually caused by an invalid file name or denied access.

Remarks

The database is created with information provided in two sets of parameters.

The connParms parameter is a set of standard connection parameters that are applicable whenever the database is accessed, such as the file name or the encryption key.

The createParms parameter is a set of parameters that are only relevant when creating a database, such as checksum-level, page-size, collation, and time and date format.

The following code illustrates how to use the CreateDatabase method to create an UltraLite database as the file `mydb.udb`:

```
ULConnection * conn;
conn = ULDatabaseManager::CreateDatabase( "DBF=mydb.udb", "kdf_iterations=100" );
if( conn != NULL ) {
   // success
} else {
```

```
    // unable to create
}
```

1.2.2 DropDatabase(const char *, ULError *) Method

Erases an existing database that is not currently running.

Syntax

```
public static bool DropDatabase (
    const char * parms,
    ULError * error
)
```

Parameters

parms The database identification parameters. (a connection string)
error An optional ULError object to receive error information.

Returns

True if the database was successfully deleted; otherwise, returns false.

1.2.3 EnableAesDBEncryption() Method

Enables AES database encryption.

Syntax

```
public static void EnableAesDBEncryption ()
```

Remarks

Call this method to use AES database encryption. Use the DBKEY connection parameter to specify the encryption passphrase. You must call this method before opening the database connection.

1.2.4 EnableAesFipsDBEncryption() Method

Enables FIPS 140-2 certified AES database encryption.

Syntax

```
public static void EnableAesFipsDBEncryption ()
```

Remarks

Call this method to use FIPS AES database encryption. Use the DBKEY connection parameter to specify the encryption passphrase.

You must specify 'fips=yes' in the database creation parameters string. You must call this method before opening the database connection.

Related Information

EnableAesDBEncryption() Method [page 50]

1.2.5 EnableAllSynchronization() Method

Enables all four types of synchronization: TCPIP, HTTP, TLS, and HTTPS.

Syntax

```
public static void EnableAllSynchronization ()
```

Remarks

You must call this method before the Synchronize method.

When initiating synchronization, set the *stream* parameter to "TCPIP", "HTTP", "TLS", or "HTTPS". Also set the network protocol certificate options if using TLS or HTTPS

1.2.6 EnableHttpsSynchronization() Method

Enables HTTPS synchronization.

Syntax

```
public static void EnableHttpsSynchronization ()
```

Remarks

You must call this method before the Synchronize method.

When initiating synchronization, set the *stream* parameter to "HTTPS". Also set the network protocol certificate options.

1.2.7 EnableHttpSynchronization() Method

Enables HTTP synchronization.

Syntax

```
public static void EnableHttpSynchronization ()
```

Remarks

You must call this method before the Synchronize method.

When initiating synchronization, set the *stream* parameter to "HTTP".

1.2.8 EnableRsaE2ee() Method

Enables RSA end-to-end encryption.

Syntax

```
public static void EnableRsaE2ee ()
```

Remarks

You must call this method before the Synchronize method.

To use end-to-end encryption, set the *e2ee_public_key* network protocol option.

1.2.9 EnableRsaFipsE2ee() Method

Enables FIPS 140-2 certified RSA end-to-end encryption.

Syntax

```
public static void EnableRsaFipsE2ee ()
```

Remarks

You must call this method before the Synchronize method.

To use end-to-end encryption, set the *e2ee_public_key* network protocol option. In this case, the *fips* option must be set to "yes".

1.2.10 EnableRsaFipsSyncEncryption() Method

Enables FIPS 140-2 certified RSA synchronization encryption for SSL or TLS streams.

Syntax

```
public static void EnableRsaFipsSyncEncryption ()
```

Remarks

You must call this method before the Synchronize method.

This is required when setting the *stream* parameter to "TLS" or "HTTPS" for FIPS RSA encryption. In this case, the *fips* option must be set to "yes".

Related Information

EnableRsaSyncEncryption() Method [page 54]

1.2.11 EnableRsaSyncEncryption() Method

Enables RSA synchronization encryption.

Syntax

```
public static void EnableRsaSyncEncryption ()
```

Remarks

You must call this method before the Synchronize method.

This is required when setting the *stream* parameter to "TLS" or "HTTPS" for RSA encryption.

1.2.12 EnableTcpipSynchronization() Method

Enables TCP/IP synchronization.

Syntax

```
public static void EnableTcpipSynchronization ()
```

Remarks

You must call this method before the Synchronize method.

When initiating synchronization, set the *stream* parameter to "TCPIP".

1.2.13 EnableTlsSynchronization() Method

Enables TLS synchronization.

Syntax

```
public static void EnableTlsSynchronization ()
```

Remarks

You must call this method before the Synchronize method.

When initiating synchronization, set the *stream* parameter to "TLS". Also set the network protocol certificate options.

1.2.14 EnableZlibSyncCompression() Method

Enables Zlib compression for a synchronization stream.

Syntax

```
public static void EnableZlibSyncCompression ()
```

Remarks

You must call this method before the Synchronize method.

To use compression, set the *compression* network protocol option to "zlib".

1.2.15 Fini() Method

Finalizes the UltraLite runtime.

Syntax

```
public static void Fini ()
```

Remarks

This method must be called only once by a single thread when the application is finished. This method is not thread-safe.

1.2.16 Init() Method

Initializes the UltraLite runtime.

Syntax

```
public static bool Init ()
```

Returns

True on success; otherwise, returns false. False can also be returned if the method is called more than once.

Remarks

This method must be called only once by a single thread before any other calls can be made. This method is not thread-safe.

This method does not usually fail unless memory is unavailable.

1.2.17 OpenConnection(const char *, ULError *, void *) Method

Opens a new connection to an existing database.

Syntax

```
public static ULConnection * OpenConnection (
    const char * connParms,
    ULError * error,
    void * reserved
)
```

Parameters

connParms The connection string.
error An optional ULError object to return error information.
reserved Reserved for internal use. Omit or set to null.

Returns

A new ULConnection object if the method succeeds; otherwise, returns NULL.

Remarks

The connection string is a set of option=value connection parameters (semicolon separated) that indicates which database to connect to, and options to use for the connection. For example, after securely obtaining your encryption passphrase, the resulting connection string might be:
"DBF=mydb.udb;DBKEY=iyntTZId9OEa#&G".

To get error information, pass in a pointer to a ULError object. The following is a list of possible errors:

SQLE_INVALID_PARSE_PARAMETER

connParms was not formatted properly.
SQLE_UNRECOGNIZED_OPTION

A connection option name was likely misspelled.
SQLE_INVALID_OPTION_VALUE

A connection option value was not specified properly.
SQLE_ULTRALITE_DATABASE_NOT_FOUND

The specified database could not be found.
SQLE_INVALID_LOGON

You supplied an invalid user ID or an incorrect password.
SQLE_TOO_MANY_CONNECTIONS

You exceeded the maximum number of concurrent database connections.

1.2.18 SetErrorCallback(ul_cpp_error_callback_fn, void *) Method

Sets the callback to be invoked when an error occurs.

Syntax

```
public static void SetErrorCallback (
```

```
    ul_cpp_error_callback_fn callback,
    void * userData
)
```

Parameters

callback The callback function.
userData User context information passed to the callback.

Remarks

This method is not thread-safe.

1.2.19 ValidateDatabase(const char *, ul_u_short, ul_validate_callback_fn, void *, ULError *) Method

Performs low level and index validation on a database.

Syntax

```
public static bool ValidateDatabase (
    const char * connParms,
    ul_u_short flags,
    ul_validate_callback_fn fn,
    void * userData,
    ULError * error
)
```

Parameters

connParms The parameters used to connect to the database.
flags The flags controlling the type of validation; see the example below.
fn A function to receive validation progress information.
userData The user data to send back to the caller via the callback.
error An optional ULError object to receive error information.

Returns

True if the validation succeeds; otherwise, returns false.

Remarks

The flags parameter is combination of the following values:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Example

The following example demonstrates table and index validation in express mode:

```
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

1.3 ULDatabaseSchema Class

Represents the schema of an UltraLite database.

Syntax

```
public class ULDatabaseSchema
```

Members

All members of ULDatabaseSchema, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual void	Close() [page 60]	Destroys this object.
public virtual ULConnection *	GetConnection() [page 61]	Gets the ULConnection object.

Modifier and Type	Method	Description
public virtual const char *	GetNextPublication(ul_publication_iter *) [page 61]	Gets the name of the next publication in the database.
public virtual ULTableSchema *	GetNextTable(ul_table_iter *) [page 62]	Gets the next table (schema) in the database.
public virtual ul_publication_count	GetPublicationCount() [page 62]	Gets the number of publications in the database.
public virtual ul_table_num	GetTableCount() [page 63]	Returns the number of tables in the database.
public virtual ULTableSchema *	GetTableSchema(const char *) [page 63]	Returns the schema of the named table.

In this section:

1.3.1 Close() Method

Destroys this object.

Syntax

```
public virtual void Close ()
```

1.3.2 GetConnection() Method

Gets the ULConnection object.

Syntax

```
public virtual ULConnection * GetConnection ()
```

Returns

The ULConnection associated with this object.

1.3.3 GetNextPublication(ul_publication_iter *) Method

Gets the name of the next publication in the database.

Syntax

```
public virtual const char * GetNextPublication (ul_publication_iter * iter)
```

Parameters

iter A pointer to the iterator variable.

Returns

The name of the next publication. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it. NULL is returned when the iteration is complete.

Remarks

Initialize the iter value to the ul_publication_iter_start constant before the first call.

Related Information

ul_publication_iter_start Variable [page 183]

1.3.4 GetNextTable(ul_table_iter *) Method

Gets the next table (schema) in the database.

Syntax

```
public virtual ULTableSchema * GetNextTable (ul_table_iter * iter)
```

Parameters

iter A pointer to the iterator variable.

Returns

A ULTableSchema object or NULL when the iteration is complete.

Remarks

Initialize the iter value to the ul_table_iter_start constant before the first call.

Related Information

ul_table_iter_start Variable [page 183]

1.3.5 GetPublicationCount() Method

Gets the number of publications in the database.

Syntax

```
public virtual ul_publication_count GetPublicationCount ()
```

Returns

The number of publications in the database.

Remarks

Publication IDs range from 1 to the number returned by this method.

1.3.6 GetTableCount() Method

Returns the number of tables in the database.

Syntax

```
public virtual ul_table_num GetTableCount ()
```

Returns

An integer that represents the number of tables.

1.3.7 GetTableSchema(const char *) Method

Returns the schema of the named table.

Syntax

```
public virtual ULTableSchema * GetTableSchema (const char * tableName)
```

Parameters

tableName The name of the table.

Returns

A ULTableSchema object for the given table; otherwise, returns UL_NULL if the table does not exist.

1.4 ULError Class

Manages the errors returned from the UltraLite runtime.

Syntax

```
public class ULError
```

Members

All members of ULError, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	ULError() [page 65]	Constructs a ULError object.

Methods

Modifier and Type	Method	Description
public void	Clear() [page 65]	Clears the current error.
public const ul_error_info *	GetErrorInfo [page 66]	Returns a pointer to the underlying ul_error_info object.
public size_t	GetParameter(ul_u_short, char *, size_t) [page 67]	Copies the specified error parameter into the provided buffer.
public ul_u_short	GetParameterCount() [page 68]	Returns the number of error parameters.
public an_sql_code	GetSQLCode() [page 68]	Returns the SQLCODE error code for the last operation.
public ul_s_long	GetSQLCount() [page 68]	Returns a value that depends on the last operation, and the result of that operation.
public size_t	GetString(char *, size_t) [page 69]	Returns the description of the current error.
public size_t	GetURL(char *, size_t, const char *) [page 70]	Returns a URL to the documentation page for this error.
public bool	IsOK() [page 70]	Tests the error code.

In this section:

1.4.1 ULError() Constructor

Constructs a ULError object.

Syntax

```
public ULError ()
```

1.4.2 Clear() Method

Clears the current error.

Syntax

```
public void Clear ()
```

Remarks

The current error is cleared automatically on most calls, so this is not normally called by applications.

1.4.3 GetErrorInfo Method

Returns a pointer to the underlying ul_error_info object.

Overload list

Modifier and Type	Overload name	Description
public const ul_error_info *	GetErrorInfo() [page 66]	Returns a pointer to the underlying ul_error_info object.
public ul_error_info *	GetErrorInfo() [page 67]	Returns a pointer to the underlying ul_error_info object.

In this section:

GetErrorInfo() Method [page 66]
Returns a pointer to the underlying ul_error_info object.

GetErrorInfo() Method [page 67]
Returns a pointer to the underlying ul_error_info object.

1.4.3.1 GetErrorInfo() Method

Returns a pointer to the underlying ul_error_info object.

Syntax

```
public const ul_error_info * GetErrorInfo ()
```

Returns

A pointer to the underlying ul_error_info object.

1.4.3.2 GetErrorInfo() Method

Returns a pointer to the underlying ul_error_info object.

Syntax

```
public ul_error_info * GetErrorInfo ()
```

Returns

A pointer to the underlying ul_error_info object.

1.4.4 GetParameter(ul_u_short, char *, size_t) Method

Copies the specified error parameter into the provided buffer.

Syntax

```
public size_t GetParameter (
    ul_u_short parmNo,
    char * dst,
    size_t len
)
```

Parameters

parmNo A 1-based parameter number.
dst The buffer to receive the parameter.
len The size of the buffer.

Returns

The size required to store the parameter, or zero if the ordinal is not valid. The parameter is truncated if the return value is larger than the len value.

Remarks

The output string is always null-terminated, even when the buffer is too small and the string is truncated.

1.4.5 GetParameterCount() Method

Returns the number of error parameters.

Syntax

```
public ul_u_short GetParameterCount ()
```

Returns

The number of error parameters.

1.4.6 GetSQLCode() Method

Returns the SQLCODE error code for the last operation.

Syntax

```
public an_sql_code GetSQLCode ()
```

Returns

The sqlcode value.

1.4.7 GetSQLCount() Method

Returns a value that depends on the last operation, and the result of that operation.

Syntax

```
public ul_s_long GetSQLCount ()
```

Returns

The value for the last operation, if applicable; otherwise, returns -1 if not applicable.

Remarks

The following list outlines the possible operations, and their returned results:

INSERT, UPDATE, or DELETE operation executed successfully

Returns the number of rows that were affected by the statement.

SQL statement syntax error (SQLE_SYNTAX_ERROR)

Returns the approximate character position within the statement where the error was detected.

1.4.8 GetString(char *, size_t) Method

Returns the description of the current error.

Syntax

```
public size_t GetString (
    char * dst,
    size_t len
)
```

Parameters

dst The buffer to receive the error description.

len The size, in array elements, of the buffer.

Returns

The size required to store the string. The string is truncated when the return value is larger than the len value.

Remarks

The string includes the error code and all parameters. A full description of the error can be obtained by loading the URL returned by the ULError.GetURL method.

The output string is always null-terminated, even if the buffer is too small and the string is truncated.

Related Information

GetURL(char *, size_t, const char *) Method [page 70]

1.4.9 GetURL(char *, size_t, const char *) Method

Returns a URL to the documentation page for this error.

Syntax

```
public size_t GetURL (
    char * buffer,
    size_t len,
    const char * reserved
)
```

Parameters

buffer The buffer to receive the URL.
len The size of the buffer.
reserved Reserved for future use; you must pass NULL, the default.

Returns

The size required to store the URL. The URL is truncated if the return value is larger is larger than the len value.

1.4.10 IsOK() Method

Tests the error code.

Syntax

```
public bool IsOK ()
```

Returns

True if the current code is SQLE_NOERROR or a warning; otherwise, returns false if the current code indicates an error.

1.5 ULIndexSchema Class

Represents the schema of an UltraLite table index.

Syntax

```
public class ULIndexSchema
```

Members

All members of ULIndexSchema, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual void	Close() [page 72]	Destroys this object.
public virtual ul_column_num	GetColumnCount() [page 72]	Gets the number of columns in the index.
public virtual const char *	GetColumnName(ul_column_num) [page 73]	Gets the name of the column given the position of the column in the index.
public virtual ULConnection *	GetConnection() [page 73]	Gets the ULConnection object.
public virtual ul_column_num	GetIndexColumnID(const char *) [page 74]	Gets the 1-based index column ID from its name.
public virtual ul_index_flag	GetIndexFlags() [page 74]	Gets the index property flags bit field.
public virtual const char *	GetName() [page 74]	Gets the name of the index.
public virtual const char *	GetReferencedIndexName() [page 75]	Gets the associated primary index name.
public virtual const char *	GetReferencedTableName() [page 75]	Gets the associated primary table name.
public virtual const char *	GetTableName() [page 76]	Gets the name of the table containing this index.
public virtual bool	IsColumnDescending(ul_column_num) [page 76]	Determines if the column is in descending order.

In this section:

1.5.1 Close() Method

Destroys this object.

Syntax

```
public virtual void Close ()
```

1.5.2 GetColumnCount() Method

Gets the number of columns in the index.

Syntax

```
public virtual ul_column_num GetColumnCount ()
```

Returns

The number of columns in the index.

1.5.3 GetColumnName(ul_column_num) Method

Gets the name of the column given the position of the column in the index.

Syntax

```
public virtual const char * GetColumnName (ul_column_num col_id_in_index)
```

Parameters

col_id_in_index The 1-based ordinal number indicating the position of the column in the index.

Returns

The name of the column. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

1.5.4 GetConnection() Method

Gets the ULConnection object.

Syntax

```
public virtual ULConnection * GetConnection ()
```

Returns

The connection associated with this object.

1.5.5 GetIndexColumnID(const char *) Method

Gets the 1-based index column ID from its name.

Syntax

```
public virtual ul_column_num GetIndexColumnID (const char * columnName)
```

Parameters

columnName The column name.

Returns

0, and sets SQLE_COLUMN_NOT_FOUND if the column name does not exist.

1.5.6 GetIndexFlags() Method

Gets the index property flags bit field.

Syntax

```
public virtual ul_index_flag GetIndexFlags ()
```

Related Information

ul_index_flag Enumeration [page 180]

1.5.7 GetName() Method

Gets the name of the index.

Syntax

```
public virtual const char * GetName ()
```

Returns

The name of the index. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

1.5.8 GetReferencedIndexName() Method

Gets the associated primary index name.

Syntax

```
public virtual const char * GetReferencedIndexName ()
```

Returns

The name of the referenced index. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

Remarks

This method applies to foreign keys only.

1.5.9 GetReferencedTableName() Method

Gets the associated primary table name.

Syntax

```
public virtual const char * GetReferencedTableName ()
```

Returns

The name of the referenced table. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

Remarks

This method applies to foreign keys only.

1.5.10 GetTableName() Method

Gets the name of the table containing this index.

Syntax

```
public virtual const char * GetTableName ()
```

Returns

The name of the table containing this index. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

1.5.11 IsColumnDescending(ul_column_num) Method

Determines if the column is in descending order.

Syntax

```
public virtual bool IsColumnDescending (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

True if the column is in descending order; otherwise, returns false.

1.6 ULPreparedStatement Class

Represents a prepared SQL statement.

Syntax

```
public class ULPreparedStatement
```

Members

All members of ULPreparedStatement, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual bool	AppendParameterByteChunk(ul_column_num, const ul_byte *, size_t) [page 79]	Sets a large binary parameter broken down into several chunks.
public virtual bool	AppendParameterStringChunk(ul_column_num, const char *, size_t) [page 80]	Sets a large string parameter broken down into several chunks.
public virtual void	Close() [page 81]	Destroys this object.
public virtual ULResultSet *	ExecuteQuery() [page 81]	Executes a SQL SELECT statement as a query.
public virtual bool	ExecuteStatement() [page 81]	Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.
public virtual ULConnection *	GetConnection() [page 82]	Gets the connection object.
public virtual ul_u_short	GetParameterCount() [page 82]	Gets the number of input parameters for this statement.
public virtual ul_column_num	GetParameterID(const char *) [page 82]	Get the 1-based ordinal for a parameter name.
public virtual ul_column_storage_type	GetParameterType(ul_column_num) [page 83]	Gets the storage/host variable type of a parameter.
public virtual size_t	GetPlan(char *, size_t) [page 83]	Gets a text-based description of the query execution plan.
public virtual const ULResultSetSchema &	GetResultSetSchema() [page 84]	Gets the schema for the result set.
public virtual ul_s_long	GetRowsAffectedCount() [page 84]	Gets the number of rows affected by the last statement.
public virtual bool	HasResultSet() [page 85]	Determines if the SQL statement has a result set.

Modifier and Type	Method	Description
public virtual bool	SetParameterBinary(ul_column_num, const p_ul_binary) [page 85]	Sets a parameter to a ul_binary value.
public virtual bool	SetParameterDateTime(ul_column_num, DECL_DATETIME *) [page 86]	Sets a parameter to a DECL_DATETIME value.
public virtual bool	SetParameterDouble(ul_column_num, ul_double) [page 87]	Sets a parameter to a double value.
public virtual bool	SetParameterFloat(ul_column_num, ul_real) [page 87]	Sets a parameter to a float value.
public virtual bool	SetParameterGuid(ul_column_num, GUID *) [page 88]	Sets a parameter to a GUID value.
public virtual bool	SetParameterInt(ul_column_num, ul_s_long) [page 89]	Sets a parameter to an integer value.
public virtual bool	SetParameterIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) [page 89]	Sets a parameter to an integer value of the specified integer type.
public virtual bool	SetParameterNull(ul_column_num) [page 90]	Sets a parameter to null.
public virtual bool	SetParameterString(ul_column_num, const char *, size_t) [page 91]	Sets a parameter to a string value.

In this section:

AppendParameterByteChunk(ul_column_num, const ul_byte *, size_t) Method [page 79]
Sets a large binary parameter broken down into several chunks.

AppendParameterStringChunk(ul_column_num, const char *, size_t) Method [page 80]
Sets a large string parameter broken down into several chunks.

Close() Method [page 81]
Destroys this object.

ExecuteQuery() Method [page 81]
Executes a prepared statement that returns a result set.

ExecuteStatement() Method [page 81]
Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.

GetConnection() Method [page 82]
Gets the connection object.

GetParameterCount() Method [page 82]
Gets the number of input parameters for the prepared statement.

GetParameterID(const char *) Method [page 82]
Get the 1-based ordinal for a parameter name.

GetParameterType(ul_column_num) Method [page 83]
Gets the storage/host variable type of a parameter.

GetPlan(char *, size_t) Method [page 83]
Gets a text-based description of the query execution plan.

GetResultSetSchema() Method [page 84]
Gets the schema for the result set.

GetRowsAffectedCount() Method [page 84]
Gets the number of rows affected by the last statement.

HasResultSet() Method [page 85]
Determines if the SQL statement has a result set.

SetParameterBinary(ul_column_num, const p_ul_binary) Method [page 85]
Sets a parameter to a ul_binary value.

SetParameterBytes(ul_column_num pid, const ul_byte * value, size_t len) Method [page 86]
Sets a parameter to a byte array value.

SetParameterDateTime(ul_column_num, DECL_DATETIME *) Method [page 86]
Sets a parameter to a DECL_DATETIME value.

SetParameterDouble(ul_column_num, ul_double) Method [page 87]
Sets a parameter to a double value.

SetParameterFloat(ul_column_num, ul_real) Method [page 87]
Sets a parameter to a float value.

SetParameterGuid(ul_column_num, GUID *) Method [page 88]
Sets a parameter to a GUID value.

SetParameterInt(ul_column_num, ul_s_long) Method [page 89]
Sets a parameter to an integer value.

SetParameterIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) Method [page 89]
Sets a parameter to an integer value of the specified integer type.

SetParameterNull(ul_column_num) Method [page 90]
Sets a parameter to null.

SetParameterString(ul_column_num, const char *, size_t) Method [page 91]
Sets a parameter to a string value.

1.6.1 AppendParameterByteChunk(ul_column_num, const ul_byte *, size_t) Method

Sets a large binary parameter broken down into several chunks.

Syntax

```
public virtual bool AppendParameterByteChunk (
    ul_column_num pid,
    const ul_byte * value,
    size_t valueSize
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The byte chunk to append.
valueSize The size of the buffer.

Returns

True on success; otherwise, returns false.

1.6.2 AppendParameterStringChunk(ul_column_num, const char *, size_t) Method

Sets a large string parameter broken down into several chunks.

Syntax

```
public virtual bool AppendParameterStringChunk (
    ul_column_num pid,
    const char * value,
    size_t len
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The string chunk to append.
len Optional. Set to the length of the string chunk in bytes or UL_NULL_TERMINATED_STRING if the string chunk is null-terminated.

Returns

True on success; otherwise, returns false.

1.6.3 Close() Method

Destroys this object.

Syntax

```
public virtual void Close ()
```

1.6.4 ExecuteQuery() Method

Executes a prepared statement that returns a result set.

Syntax

```
public virtual ULResultSet * ExecuteQuery ()
```

Returns

The ULResultSet object that contains the results of the query, as a set of rows.

1.6.5 ExecuteStatement() Method

Executes a statement that does not return a result set, such as a SQL INSERT, DELETE or UPDATE statement.

Syntax

```
public virtual bool ExecuteStatement ()
```

Returns

True on success; otherwise, returns false.

1.6.6 GetConnection() Method

Gets the connection object.

Syntax

```
public virtual ULConnection * GetConnection ()
```

Returns

The ULConnection object associated with this prepared statement.

1.6.7 GetParameterCount() Method

Gets the number of input parameters for the prepared statement.

Syntax

```
public virtual ul_u_short GetParameterCount ()
```

Returns

The number of input parameters for the prepared statement.

1.6.8 GetParameterID(const char *) Method

Get the 1-based ordinal for a parameter name.

Syntax

```
public virtual ul_column_num GetParameterID (const char * name)
```

Parameters

name The name of the host variable.

Returns

The 1-based ordinal for a parameter name.

1.6.9 GetParameterType(ul_column_num) Method

Gets the storage/host variable type of a parameter.

Syntax

```
public virtual ul_column_storage_type GetParameterType (ul_column_num pid)
```

Parameters

pid The 1-based ordinal of the parameter.

Returns

The type of the specified parameter.

1.6.10 GetPlan(char *, size_t) Method

Gets a text-based description of the query execution plan.

Syntax

```
public virtual size_t GetPlan (
    char * dst,
    size_t dstSize
)
```

Parameters

dst The destination buffer for the plan text. Pass NULL to determine the size of the buffer required to hold the plan.

dstSize The size of the destination buffer.

Returns

The number of bytes copied to the buffer; otherwise, if the dst value is NULL, returns the number of bytes required to store the plan, excluding the null-terminator.

Remarks

This method is intended primarily for use during development.

An empty string is returned if there is no plan. Plans exist when the prepared statement is a SQL query.

When the plan is obtained before the associated query has been executed, the plan shows the operations used to execute the query. The plan additionally shows the number of rows each operation produced when the plan is obtained after the query has been executed. This plan can be used to gain insight about the execution of the query.

1.6.11 GetResultSetSchema() Method

Gets the schema for the result set.

Syntax

```
public virtual const ULResultSetSchema & GetResultSetSchema ()
```

Returns

A ULResultSetSchema object that can be used to get information about the schema of the result set.

1.6.12 GetRowsAffectedCount() Method

Gets the number of rows affected by the last statement.

Syntax

```
public virtual ul_s_long GetRowsAffectedCount ()
```

Returns

The number of rows affected by the last statement. If the number of rows is not available (for instance, the statement alters the schema rather than data) the return value is -1.

1.6.13 HasResultSet() Method

Determines if the SQL statement has a result set.

Syntax

```
public virtual bool HasResultSet ()
```

Returns

True if a result set is generated when this statement is executed; otherwise, returns false if no result set is generated.

1.6.14 SetParameterBinary(ul_column_num, const p_ul_binary) Method

Sets a parameter to a ul_binary value.

Syntax

```
public virtual bool SetParameterBinary (
    ul_column_num pid,
    const p_ul_binary value
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The ul_binary value.

Returns

True on success; otherwise, returns false.

1.6.15 SetParameterBytes(ul_column_num pid, const ul_byte * value, size_t len) Method

Sets a parameter to a byte array value.

Syntax

```
public virtual size_t SetParameterBytes (
    ul_column_num cid,
    ul_byte * value,
    size_t len
)
```

Parameters

cid The 1-based ordinal of the parameter.
value The byte array value.
len The size of the array value.

Returns

True on success; otherwise, returns false.

1.6.16 SetParameterDateTime(ul_column_num, DECL_DATETIME *) Method

Sets a parameter to a DECL_DATETIME value.

Syntax

```
public virtual bool SetParameterDateTime (
    ul_column_num pid,
    DECL_DATETIME * value
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The DECL_DATETIME value.

Returns

True on success; otherwise, returns false.

1.6.17 SetParameterDouble(ul_column_num, ul_double) Method

Sets a parameter to a double value.

Syntax

```
public virtual bool SetParameterDouble (
    ul_column_num pid,
    ul_double value
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The double value.

Returns

True on success; otherwise, returns false.

1.6.18 SetParameterFloat(ul_column_num, ul_real) Method

Sets a parameter to a float value.

Syntax

```
public virtual bool SetParameterFloat (
```

```
    ul_column_num pid,
    ul_real value
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The float value.

Returns

True on success; otherwise, returns false.

1.6.19 SetParameterGuid(ul_column_num, GUID *) Method

Sets a parameter to a GUID value.

Syntax

```
public virtual bool SetParameterGuid (
    ul_column_num pid,
    GUID * value
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The GUID value.

Returns

True on success; otherwise, returns false.

1.6.20 SetParameterInt(ul_column_num, ul_s_long) Method

Sets a parameter to an integer value.

Syntax

```
public virtual bool SetParameterInt (
    ul_column_num pid,
    ul_s_long value
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The integer value.

Returns

True on success; otherwise, returns false.

1.6.21 SetParameterIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) Method

Sets a parameter to an integer value of the specified integer type.

Syntax

```
public virtual bool SetParameterIntWithType (
    ul_column_num pid,
    ul_s_big value,
    ul_column_storage_type type
)
```

Parameters

pid The 1-based ordinal of the parameter.
value The integer value.
type The integer type to treat the value as.

Returns

True on success; otherwise, returns false.

Remarks

The following is a list of integer values that can be used for the value parameter:

- UL_TYPE_BIT
- UL_TYPE_TINY
- UL_TYPE_S_SHORT
- UL_TYPE_U_SHORT
- UL_TYPE_S_LONG
- UL_TYPE_U_LONG
- UL_TYPE_S_BIG
- UL_TYPE_U_BIG

1.6.22 SetParameterNull(ul_column_num) Method

Sets a parameter to null.

Syntax

```
public virtual bool SetParameterNull (ul_column_num pid)
```

Parameters

pid The 1-based ordinal of the parameter.

Returns

True on success; otherwise, returns false.

1.6.23 SetParameterString(ul_column_num, const char *, size_t) Method

Sets a parameter to a string value.

Syntax

```
public virtual bool SetParameterString (
    ul_column_num pid,
    const char * value,
    size_t len
)
```

Parameters

pid The 1-based ordinal of the parameter.

value The string value.

len Optional. Set to the length of the string in bytes or UL_NULL_TERMINATED_STRING if the string in null-terminated. SQLE_INVALID_PARAMETER is set if this parameter is greater than 32K. For large strings, call the AppendParameterStringChunk method instead.

Returns

True on success, otherwise false.

Related Information

AppendParameterStringChunk(ul_column_num, const char *, size_t) Method [page 80]

1.7 ULResultSet Class

Represents a result set in an UltraLite database.

Syntax

```
public class ULResultSet
```

Members

All members of ULResultSet, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual bool	AfterLast() [page 95]	Moves the cursor after the last row.
public virtual bool	AppendByteChunk [page 96]	Appends bytes to a column.
public virtual bool	AppendStringChunk [page 98]	Appends a string chunk to a column.
public virtual bool	BeforeFirst() [page 100]	Moves the cursor before the first row.
public virtual void	Close() [page 100]	Destroys this object.
public virtual bool	Delete() [page 101]	Deletes the current row and moves it to the next valid row.
public virtual bool	DeleteNamed(const char *) [page 101]	Deletes the current row and moves it to the next valid row.
public virtual bool	First() [page 101]	Moves the cursor to the first row.
public virtual bool	GetBinary [page 102]	Fetches a value from a column as a ul_binary value.
public virtual size_t	GetBinaryLength [page 104]	Gets the binary length of the value of a column.
public virtual size_t	GetByteChunk [page 106]	Gets a binary chunk from the column.
public virtual ULConnection *	GetConnection() [page 108]	Gets the connection object.
public virtual bool	GetDateTime [page 109]	Fetches a value from a column as a DECL_DATETIME.
public virtual ul_double	GetDouble [page 110]	Fetches a value from a column as a double.
public virtual ul_real	GetFloat [page 112]	Fetches a value from a column as a float.
public virtual bool	GetGuid [page 114]	Fetches a value from a column as a GUID.
public virtual ul_s_long	GetInt [page 115]	Fetches a value from a column as an integer.
public virtual ul_s_big	GetIntWithType [page 117]	Fetches a value from a column as the specified integer type.
public virtual const ULResultSetSchema &	GetResultSetSchema() [page 119]	Returns an object that can be used to get information about the result set.
public virtual ul_u_long	GetRowCount(ul_u_long) [page 119]	Gets the number of rows in the table.
public virtual UL_RS_STATE	GetState() [page 120]	Gets the internal state of the cursor.
public virtual bool	GetString [page 120]	Fetches a value from a column as a null-terminated string.
public virtual size_t	GetStringChunk [page 122]	Gets a string chunk from the column.

Modifier and Type	Method	Description
public virtual size_t	GetStringLength [page 125]	Gets the string length of the value of a column.
public virtual bool	IsNull [page 127]	Checks if a column is NULL.
public virtual bool	Last() [page 129]	Moves the cursor to the last row.
public virtual bool	Next() [page 129]	Moves the cursor forward one row.
public virtual bool	Previous() [page 129]	Moves the cursor back one row.
public virtual bool	Relative(ul_fetch_offset) [page 130]	Moves the cursor by offset rows from the current cursor position.
public virtual bool	SetBinary [page 130]	Sets a column to a ul_binary value.
public virtual bool	SetDateTime [page 135]	Sets a column to a DECL_DATETIME value.
public virtual bool	SetDefault [page 137]	Sets a column to its default value.
public virtual bool	SetDouble [page 138]	Sets a column to a double value.
public virtual bool	SetFloat [page 140]	Sets a column to a float value.
public virtual bool	SetGuid [page 141]	Sets a column to a GUID value.
public virtual bool	SetInt [page 143]	Sets a column to an integer value.
public virtual bool	SetIntWithType [page 133]	Sets a column to an integer value of the specified integer type.
public virtual bool	SetNull [page 145]	Sets a column to null.
public virtual bool	SetString [page 146]	Sets a column to a string value.
public virtual bool	Update() [page 148]	Updates the current row.
public virtual bool	UpdateBegin() [page 148]	Selects the update mode for setting columns.

In this section:

AfterLast() Method [page 95]
Moves the cursor after the last row.

AppendByteChunk Method [page 96]
Appends bytes to a column.

AppendStringChunk Method [page 98]
Appends a string chunk to a column.

BeforeFirst() Method [page 100]
Moves the cursor before the first row.

Close() Method [page 100]
Destroys this object.

Delete() Method [page 101]
Deletes the current row and moves it to the next valid row.

DeleteNamed(const char *) Method [page 101]

Deletes the current row and moves it to the next valid row.

First() Method [page 101]
Moves the cursor to the first row.

GetBinary Method [page 102]
Fetches a value from a column as a ul_binary value.

GetBinaryLength Method [page 104]
Gets the binary length of the value of a column.

GetBytes(ul_column_num cid, ul_byte * dst, size_t len) Method [page 105]
Fetches a value from a column as an array of bytes.

GetByteChunk Method [page 106]
Gets a binary chunk from the column.

GetConnection() Method [page 108]
Gets the connection object.

GetDateTime Method [page 109]
Fetches a value from a column as a DECL_DATETIME.

GetDouble Method [page 110]
Fetches a value from a column as a double.

GetFloat Method [page 112]
Fetches a value from a column as a float.

GetGuid Method [page 114]
Fetches a value from a column as a GUID.

GetInt Method [page 115]
Fetches a value from a column as an integer.

GetIntWithType Method [page 117]
Fetches a value from a column as the specified integer type.

GetResultSetSchema() Method [page 119]
Returns an object that can be used to get information about the result set.

GetRowCount(ul_u_long) Method [page 119]
Gets the number of rows in the table.

GetState() Method [page 120]
Gets the internal state of the cursor.

GetString Method [page 120]
Fetches a value from a column as a null-terminated string.

GetStringChunk Method [page 122]
Gets a string chunk from the column.

GetStringLength Method [page 125]
Gets the string length of the value of a column.

IsNull Method [page 127]
Checks if a column is NULL.

Last() Method [page 129]
Moves the cursor to the last row.

1.7.1 AfterLast() Method

Moves the cursor after the last row.

Syntax

```
public virtual bool AfterLast ()
```

Returns

True on success; otherwise, returns false.

1.7.2 AppendByteChunk Method

Appends bytes to a column.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	AppendByteChunk(const char *, const ul_byte *, size_t) [page 96]	Appends bytes to a column.
public virtual bool	AppendByteChunk(ul_column_num, const ul_byte *, size_t) [page 97]	Appends bytes to a column.

In this section:

1.7.2.1 AppendByteChunk(const char *, const ul_byte *, size_t) Method

Appends bytes to a column.

Syntax

```
public virtual bool AppendByteChunk (
    const char * cname,
    const ul_byte * value,
    size_t valueSize
)
```

Parameters

cname The name of the column.

value The byte chunk to append.
valueSize The size of the byte chunk in bytes.

Returns

True on success; otherwise, returns false.

Remarks

The given bytes are appended to the end of the column written so far by AppendBinaryChunk method calls.

Related Information

AppendByteChunk(ul_column_num, const ul_byte *, size_t) Method [page 97]

1.7.2.2 AppendByteChunk(ul_column_num, const ul_byte *, size_t) Method

Appends bytes to a column.

Syntax

```
public virtual bool AppendByteChunk (
    ul_column_num cid,
    const ul_byte * value,
    size_t valueSize
)
```

Parameters

cid The 1-based ordinal column number.
value The byte chunk to append.
valueSize The size of the byte chunk in bytes.

Returns

True on success; otherwise, returns false.

Remarks

The given bytes are appended to the end of the column written so far by AppendBinaryChunk method calls.

1.7.3 AppendStringChunk Method

Appends a string chunk to a column.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	AppendStringChunk(const char *, const char *, size_t) [page 98]	Appends a string chunk to a column.
public virtual bool	AppendStringChunk(ul_column_num, const char *, size_t) [page 99]	Appends a string chunk to a column.

In this section:

AppendStringChunk(const char *, const char *, size_t) Method [page 98]
Appends a string chunk to a column.

AppendStringChunk(ul_column_num, const char *, size_t) Method [page 99]
Appends a string chunk to a column.

1.7.3.1 AppendStringChunk(const char *, const char *, size_t) Method

Appends a string chunk to a column.

Syntax

```
public virtual bool AppendStringChunk (
    const char * cname,
    const char * value,
    size_t len
)
```

Parameters

cname The name of the column.
value The string chunk to append.
len Optional. The length of the string chunk in bytes or the UL_NULL_TERMINATED_STRING constant if the string is null-terminated.

Returns

True on success; otherwise, returns false.

Remarks

This method appends the given string to the end of the string written so far by AppendStringChunk method calls.

Related Information

AppendStringChunk(ul_column_num, const char *, size_t) Method [page 99]

1.7.3.2 AppendStringChunk(ul_column_num, const char *, size_t) Method

Appends a string chunk to a column.

Syntax

```
public virtual bool AppendStringChunk (
    ul_column_num cid,
    const char * value,
    size_t len
)
```

Parameters

cid The 1-based ordinal column number.
value The string chunk to append.

len Optional. The length of the string chunk in bytes or the UL_NULL_TERMINATED_STRING constant if the string is null-terminated.

Returns

True on success; otherwise, returns false.

Remarks

This method appends the given string to the end of the string written so far by AppendStringChunk method calls.

1.7.4 BeforeFirst() Method

Moves the cursor before the first row.

Syntax

```
public virtual bool BeforeFirst ()
```

Returns

True on success; otherwise, returns false.

1.7.5 Close() Method

Destroys this object.

Syntax

```
public virtual void Close ()
```

1.7.6 Delete() Method

Deletes the current row and moves it to the next valid row.

Syntax

```
public virtual bool Delete ()
```

Returns

True on success, otherwise false.

1.7.7 DeleteNamed(const char *) Method

Deletes the current row and moves it to the next valid row.

Syntax

```
public virtual bool DeleteNamed (const char * tableName)
```

Parameters

tableName A table name or its correlation (required when the database has multiple columns that share the same table name).

Returns

True on success; otherwise, returns false.

1.7.8 First() Method

Moves the cursor to the first row.

Syntax

```
public virtual bool First ()
```

Returns

True on success; otherwise, returns false.

1.7.9 GetBinary Method

Fetches a value from a column as a ul_binary value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	GetBinary(const char *, p_ul_binary, size_t) [page 102]	Fetches a value from a column as a ul_binary value.
public virtual bool	GetBinary(ul_column_num, p_ul_binary, size_t) [page 103]	Fetches a value from a column as a ul_binary value.

In this section:

GetBinary(const char *, p_ul_binary, size_t) Method [page 102]
Fetches a value from a column as a ul_binary value.

GetBinary(ul_column_num, p_ul_binary, size_t) Method [page 103]
Fetches a value from a column as a ul_binary value.

1.7.9.1 GetBinary(const char *, p_ul_binary, size_t) Method

Fetches a value from a column as a ul_binary value.

Syntax

```
public virtual bool GetBinary (
    const char * cname,
    p_ul_binary dst,
    size_t len
)
```

Parameters

cname The name of the column.

dst The ul_binary result.

len The size of the ul_binary object.

Returns

True if the value was successfully fetched.

1.7.9.2 GetBinary(ul_column_num, p_ul_binary, size_t) Method

Fetches a value from a column as a ul_binary value.

Syntax

```
public virtual bool GetBinary (
    ul_column_num cid,
    p_ul_binary dst,
    size_t len
)
```

Parameters

cid The 1-based ordinal column number.

dst The ul_binary result.

len The size of the ul_binary object.

Returns

True if the value was successfully fetched.

1.7.10 GetBinaryLength Method

Gets the binary length of the value of a column.

Overload list

Modifier and Type	Overload name	Description
public virtual size_t	GetBinaryLength(const char *) [page 104]	Gets the binary length of the value of a column.
public virtual size_t	GetBinaryLength(ul_column_num) [page 105]	Gets the binary length of the value of a column.

In this section:

1.7.10.1 GetBinaryLength(const char *) Method

Gets the binary length of the value of a column.

Syntax

```
public virtual size_t GetBinaryLength (const char * cname)
```

Parameters

cname The name of the column.

Returns

The size of the column value as a binary

1.7.10.2 GetBinaryLength(ul_column_num) Method

Gets the binary length of the value of a column.

Syntax

```
public virtual size_t GetBinaryLength (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

The size of the column value as a binary

1.7.11 GetBytes(ul_column_num cid, ul_byte * dst, size_t len) Method

Fetches a value from a column as an array of bytes.

Syntax

```
public virtual size_t GetBytes (
    ul_column_num cid,
    ul_byte * dst,
    size_t len
)
```

Parameters

cid The 1-based ordinal column number.
dst The buffer to hold the bytes.
len The size of the buffer in bytes.

Returns

The number of bytes copied to the destination buffer. 0 is returned if the value is NULL.

1.7.12 GetByteChunk Method

Gets a binary chunk from the column.

Overload list

Modifier and Type	Overload name	Description
public virtual size_t	GetByteChunk(const char *, ul_byte *, size_t, size_t) [page 106]	Gets a binary chunk from the column.
public virtual size_t	GetByteChunk(ul_column_num, ul_byte *, size_t, size_t) [page 107]	Gets a binary chunk from the column.

In this section:

1.7.12.1 GetByteChunk(const char *, ul_byte *, size_t, size_t) Method

Gets a binary chunk from the column.

Syntax

```
public virtual size_t GetByteChunk (
    const char * cname,
    ul_byte * dst,
    size_t len,
    size_t offset
)
```

Parameters

cname The name of the column.
dst The buffer to hold the bytes.
len The size of the buffer in bytes.
offset The offset into the value at which to start reading or the UL_BLOB_CONTINUE constant to continue from where the last read ended.

Returns

The number of bytes copied to the destination buffer. If the dst value is NULL, then the number of bytes left is returned. An empty string is returned in the dst parameter when the column is null; use the IsNull method to differentiate between null and empty strings.

Remarks

The end of the value has been reached if 0 is returned.

Related Information

IsNull(ul_column_num) Method [page 128]

1.7.12.2 GetByteChunk(ul_column_num, ul_byte *, size_t, size_t) Method

Gets a binary chunk from the column.

Syntax

```
public virtual size_t GetByteChunk (
    ul_column_num cid,
    ul_byte * dst,
    size_t len,
    size_t offset
)
```

Parameters

cid The 1-based ordinal column number.
dst The buffer to hold the bytes.
len The size of the buffer in bytes.
offset The offset into the value at which to start reading or the UL_BLOB_CONTINUE constant to continue from where the last read ended.

Returns

The number of bytes copied to the destination buffer. If the dst value is NULL, then the number of bytes left is returned. An empty string is returned in the dst parameter when the column is null; use the IsNull method to differentiate between null and empty strings.

Remarks

The end of the value has been reached if 0 is returned.

Related Information

IsNull(ul_column_num) Method [page 128]

1.7.13 GetConnection() Method

Gets the connection object.

Syntax

```
public virtual ULConnection * GetConnection ()
```

Returns

The ULConnection object associated with this result set.

1.7.14 GetDateTime Method

Fetches a value from a column as a DECL_DATETIME.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	GetDateTime(const char *, DECL_DATETIME *) [page 109]	Fetches a value from a column as a DECL_DATETIME.
public virtual bool	GetDateTime(ul_column_num, DECL_DATETIME *) [page 110]	Fetches a value from a column as a DECL_DATETIME.

In this section:

GetDateTime(const char *, DECL_DATETIME *) Method [page 109]
Fetches a value from a column as a DECL_DATETIME.

GetDateTime(ul_column_num, DECL_DATETIME *) Method [page 110]
Fetches a value from a column as a DECL_DATETIME.

1.7.14.1 GetDateTime(const char *, DECL_DATETIME *) Method

Fetches a value from a column as a DECL_DATETIME.

Syntax

```
public virtual bool GetDateTime (
    const char * cname,
    DECL_DATETIME * dst
)
```

Parameters

cname The name of the column.
dst The DECL_DATETIME value.

Returns

True if the value was successfully fetched.

1.7.14.2 GetDateTime(ul_column_num, DECL_DATETIME *) Method

Fetches a value from a column as a DECL_DATETIME.

Syntax

```
public virtual bool GetDateTime (
    ul_column_num cid,
    DECL_DATETIME * dst
)
```

Parameters

cid The 1-based ordinal column number.
dst The DECL_DATETIME value.

Returns

True if the value was successfully fetched.

1.7.15 GetDouble Method

Fetches a value from a column as a double.

Overload list

Modifier and Type	Overload name	Description
public virtual ul_double	GetDouble(const char *) [page 111]	Fetches a value from a column as a double.

Modifier and Type	Overload name	Description
public virtual ul_double	GetDouble(ul_column_num) [page 111]	Fetches a value from a column as a double.

In this section:

1.7.15.1 GetDouble(const char *) Method

Fetches a value from a column as a double.

Syntax

```
public virtual ul_double GetDouble (const char * cname)
```

Parameters

cname The name of the column.

Returns

The column value as a double.

1.7.15.2 GetDouble(ul_column_num) Method

Fetches a value from a column as a double.

Syntax

```
public virtual ul_double GetDouble (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

The column value as a double.

1.7.16 GetFloat Method

Fetches a value from a column as a float.

Overload list

Modifier and Type	Overload name	Description
public virtual ul_real	GetFloat(const char *) [page 112]	Fetches a value from a column as a float.
public virtual ul_real	GetFloat(ul_column_num) [page 113]	Fetches a value from a column as a float.

In this section:

GetFloat(const char *) Method [page 112]
Fetches a value from a column as a float.

GetFloat(ul_column_num) Method [page 113]
Fetches a value from a column as a float.

1.7.16.1 GetFloat(const char *) Method

Fetches a value from a column as a float.

Syntax

```
public virtual ul_real GetFloat (const char * cname)
```

Parameters

cname The name of the column.

Returns

The column value as a float.

1.7.16.2 GetFloat(ul_column_num) Method

Fetches a value from a column as a float.

Syntax

```
public virtual ul_real GetFloat (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

The column value as a float.

1.7.17 GetGuid Method

Fetches a value from a column as a GUID.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	GetGuid(const char *, GUID *) [page 114]	Fetches a value from a column as a GUID.
public virtual bool	GetGuid(ul_column_num, GUID *) [page 115]	Fetches a value from a column as a GUID.

In this section:

1.7.17.1 GetGuid(const char *, GUID *) Method

Fetches a value from a column as a GUID.

Syntax

```
public virtual bool GetGuid (
    const char * cname,
    GUID * dst
)
```

Parameters

cname The name of the column.
dst The GUID value.

Returns

True if the value was successfully fetched.

1.7.17.2 GetGuid(ul_column_num, GUID *) Method

Fetches a value from a column as a GUID.

Syntax

```
public virtual bool GetGuid (
    ul_column_num cid,
    GUID * dst
)
```

Parameters

cid The 1-based ordinal column number.
dst The GUID value.

Returns

True if the value was successfully fetched.

1.7.18 GetInt Method

Fetches a value from a column as an integer.

Overload list

Modifier and Type	Overload name	Description
public virtual ul_s_long	GetInt(const char *) [page 116]	Fetches a value from a column as an integer.
public virtual ul_s_long	GetInt(ul_column_num) [page 116]	Fetches a value from a column as an integer.

In this section:

GetInt(const char *) Method [page 116]
Fetches a value from a column as an integer.

GetInt(ul_column_num) Method [page 116]
Fetches a value from a column as an integer.

1.7.18.1 GetInt(const char *) Method

Fetches a value from a column as an integer.

Syntax

```
public virtual ul_s_long GetInt (const char * cname)
```

Parameters

cname The name of the column.

Returns

The column value as an integer.

1.7.18.2 GetInt(ul_column_num) Method

Fetches a value from a column as an integer.

Syntax

```
public virtual ul_s_long GetInt (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

The column value as an integer.

1.7.19 GetIntWithType Method

Fetches a value from a column as the specified integer type.

Overload list

Modifier and Type	Overload name	Description
public virtual ul_s_big	GetIntWithType(const char *, ul_column_storage_type) [page 117]	Fetches a value from a column as the specified integer type.
public virtual ul_s_big	GetIntWithType(ul_column_num, ul_column_storage_type) [page 118]	Fetches a value from a column as the specified integer type.

In this section:

GetIntWithType(const char *, ul_column_storage_type) Method [page 117]
Fetches a value from a column as the specified integer type.

GetIntWithType(ul_column_num, ul_column_storage_type) Method [page 118]
Fetches a value from a column as the specified integer type.

1.7.19.1 GetIntWithType(const char *, ul_column_storage_type) Method

Fetches a value from a column as the specified integer type.

Syntax

```
public virtual ul_s_big GetIntWithType (
    const char * cname,
    ul_column_storage_type type
)
```

Parameters

cname The name of the column.
type The integer type to fetch as.

Returns

The column value as an integer.

Remarks

The following is a list of integer values that can be used for the type parameter:

- UL_TYPE_BIT
- UL_TYPE_TINY
- UL_TYPE_S_SHORT
- UL_TYPE_U_SHORT
- UL_TYPE_S_LONG
- UL_TYPE_U_LONG
- UL_TYPE_S_BIG
- UL_TYPE_U_BIG

1.7.19.2 GetIntWithType(ul_column_num, ul_column_storage_type) Method

Fetches a value from a column as the specified integer type.

Syntax

```
public virtual ul_s_big GetIntWithType (
    ul_column_num cid,
    ul_column_storage_type type
)
```

Parameters

cid The 1-based ordinal column number.
type The integer type to fetch as.

Returns

The column value as an integer.

Remarks

The following is a list of integer values that can be used for the type parameter:

- UL_TYPE_BIT
- UL_TYPE_TINY
- UL_TYPE_S_SHORT
- UL_TYPE_U_SHORT
- UL_TYPE_S_LONG
- UL_TYPE_U_LONG
- UL_TYPE_S_BIG
- UL_TYPE_U_BIG

1.7.20 GetResultSetSchema() Method

Returns an object that can be used to get information about the result set.

Syntax

```
public virtual const ULResultSetSchema & GetResultSetSchema ()
```

Returns

A ULResultSetSchema object that can be used to get information about the result set.

1.7.21 GetRowCount(ul_u_long) Method

Gets the number of rows in the table.

Syntax

```
public virtual ul_u_long GetRowCount (ul_u_long threshold)
```

Parameters

threshold The limit on the number of rows to count. Set to 0 to indicate no limit.

Returns

The number of rows in the table.

Remarks

This method is equivalent to executing the "SELECT COUNT(*) FROM table" statement.

1.7.22 GetState() Method

Gets the internal state of the cursor.

Syntax

```
public virtual UL_RS_STATE GetState ()
```

Returns

The state of the cursor.

1.7.23 GetString Method

Fetches a value from a column as a null-terminated string.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	GetString(const char *, char *, size_t) [page 121]	Fetches a value from a column as a null-terminated string.
public virtual bool	GetString(ul_column_num, char *, size_t) [page 121]	Fetches a value from a column as a null-terminated string.

In this section:

Fetches a value from a column as a null-terminated string.

GetString(ul_column_num, char *, size_t) Method [page 121]
Fetches a value from a column as a null-terminated string.

1.7.23.1 GetString(const char *, char *, size_t) Method

Fetches a value from a column as a null-terminated string.

Syntax

```
public virtual bool GetString (
    const char * cname,
    char * dst,
    size_t len
)
```

Parameters

cname The name of the column.
dst The buffer to hold the string value. The string is null-terminated even if truncated.
len The size of the buffer in bytes.

Returns

True if the value was successfully fetched.

Remarks

The string is truncated in the buffer when it isn't large enough to hold the entire value.

1.7.23.2 GetString(ul_column_num, char *, size_t) Method

Fetches a value from a column as a null-terminated string.

Syntax

```
public virtual bool GetString (
    ul_column_num cid,
    char * dst,
```

```
    size_t len
)
```

Parameters

cid The 1-based ordinal column number.
dst The buffer to hold the string value. The string is null-terminated even if truncated.
len The size of the buffer in bytes.

Returns

True if the value was successfully fetched.

Remarks

The string is truncated in the buffer when it isn't large enough to hold the entire value.

1.7.24 GetStringChunk Method

Gets a string chunk from the column.

Overload list

Modifier and Type	Overload name	Description
public virtual size_t	GetStringChunk(const char *, char *, size_t, size_t) [page 123]	Gets a string chunk from the column.
public virtual size_t	GetStringChunk(ul_column_num, char *, size_t, size_t) [page 124]	Gets a string chunk from the column.

In this section:

GetStringChunk(const char *, char *, size_t, size_t) Method [page 123]
Gets a string chunk from the column.

GetStringChunk(ul_column_num, char *, size_t, size_t) Method [page 124]
Gets a string chunk from the column.

1.7.24.1 GetStringChunk(const char *, char *, size_t, size_t) Method

Gets a string chunk from the column.

Syntax

```
public virtual size_t GetStringChunk (
    const char * cname,
    char * dst,
    size_t len,
    size_t offset
)
```

Parameters

cname The name of the column.

dst The buffer to hold the string chunk. The string is null-terminated even if truncated.

len The size of the buffer in bytes.

offset The offset into the value at which to start reading or the UL_BLOB_CONTINUE constant to continue from where the last read ended.

Returns

The number of bytes copied to the destination buffer excluding the null-terminator. If the dst value is set to NULL, then the number of bytes left in the string is returned. An empty string is returned in the dst parameter when the column is null; use the IsNull method to differentiate between null and empty strings.

Remarks

The end of the value has been reached if 0 is returned.

Related Information

IsNull(ul_column_num) Method [page 128]

1.7.24.2 GetStringChunk(ul_column_num, char *, size_t, size_t) Method

Gets a string chunk from the column.

Syntax

```
public virtual size_t GetStringChunk (
    ul_column_num cid,
    char * dst,
    size_t len,
    size_t offset
)
```

Parameters

cid The 1-based ordinal column number.

dst The buffer to hold the string chunk. The string is null-terminated even if truncated.

len The size of the buffer in bytes.

offset Set to the offset into the value at which to start reading or set to the UL_BLOB_CONTINUE constant to continue from where the last read ended.

Returns

The number of bytes copied to the destination buffer excluding the null-terminator. If the dst value is set to NULL, then the number of bytes left in the string is returned. An empty string is returned in the dst parameter when the column is null; use the IsNull method to differentiate between null and empty strings.

Remarks

The end of the value has been reached if 0 is returned.

Related Information

IsNull(ul_column_num) Method [page 128]

1.7.25 GetStringLength Method

Gets the string length of the value of a column.

Overload list

Modifier and Type	Overload name	Description
public virtual size_t	GetStringLength(const char *) [page 125]	Gets the string length of the value of a column.
public virtual size_t	GetStringLength(ul_column_num) [page 126]	Gets the string length of the value of a column.

In this section:

GetStringLength(const char *) Method [page 125]
Gets the string length of the value of a column.

GetStringLength(ul_column_num) Method [page 126]
Gets the string length of the value of a column.

1.7.25.1 GetStringLength(const char *) Method

Gets the string length of the value of a column.

Syntax

```
public virtual size_t GetStringLength (const char * cname)
```

Parameters

cname The name of the column.

Returns

The number of bytes or characters required to hold the string returned by one of the GetString methods, not including the null-terminator.

Remarks

The following example demonstrates how to get the string length of a column:

```
len = result_set->GetStringLength( cid );
dst = new char[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

For wide characters, the usage is as follows:

```
len = result_set->GetStringLength( cid );
dst = new ul_wchar[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

Related Information

GetString(ul_column_num, char *, size_t) Method [page 121]

1.7.25.2 GetStringLength(ul_column_num) Method

Gets the string length of the value of a column.

Syntax

```
public virtual size_t GetStringLength (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

The number of bytes or characters required to hold the string returned by one of the GetString methods, not including the null-terminator.

Remarks

The following example illustrates how to get the string length of a column:

```
len = result_set->GetStringLength( cid );
dst = new char[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

For wide characters, the usage is as follows:

```
len = result_set->GetStringLength( cid );
dst = new ul_wchar[ len + 1 ];
result_set->GetString( cid, dst, len + 1 );
```

Related Information

GetString(ul_column_num, char *, size_t) Method [page 121]

1.7.26 IsNull Method

Checks if a column is NULL.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	IsNull(const char *) [page 128]	Checks if a column is NULL.
public virtual bool	IsNull(ul_column_num) [page 128]	Checks if a column is NULL.

In this section:

IsNull(const char *) Method [page 128]
Checks if a column is NULL.

IsNull(ul_column_num) Method [page 128]
Checks if a column is NULL.

1.7.26.1 IsNull(const char *) Method

Checks if a column is NULL.

Syntax

```
public virtual bool IsNull (const char * cname)
```

Parameters

cname The name of the column.

Returns

True if the value for the column is NULL.

1.7.26.2 IsNull(ul_column_num) Method

Checks if a column is NULL.

Syntax

```
public virtual bool IsNull (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

True if the value for the column is NULL.

1.7.27 Last() Method

Moves the cursor to the last row.

Syntax

```
public virtual bool Last ()
```

Returns

True on success; otherwise, returns false.

1.7.28 Next() Method

Moves the cursor forward one row.

Syntax

```
public virtual bool Next ()
```

Returns

True, if the cursor successfully moves forward. Despite returning true, an error may be signaled even when the cursor moves successfully to the next row. For example, there could be conversion errors while evaluating the SELECT expressions. In this case, errors are also returned when retrieving the column values. False is returned if it fails to move forward. For example, there may not be a next row. In this case, the resulting cursor position is set after the last row.

1.7.29 Previous() Method

Moves the cursor back one row.

Syntax

```
public virtual bool Previous ()
```

Returns

True, if the cursor successfully moves back one row. False, if it fails to move backward. The resulting cursor position is set before the first row.

1.7.30 Relative(ul_fetch_offset) Method

Moves the cursor by offset rows from the current cursor position.

Syntax

```
public virtual bool Relative (ul_fetch_offset offset)
```

Parameters

offset The number of rows to move.

Returns

True on success; otherwise, returns false.

1.7.31 SetBinary Method

Sets a column to a ul_binary value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetBinary(const char *, p_ul_binary) [page 131]	Sets a column to a ul_binary value.
public virtual bool	SetBinary(ul_column_num, p_ul_binary) [page 131]	Sets a column to a ul_binary value.

In this section:

SetBinary(const char *, p_ul_binary) Method [page 131]

Sets a column to a ul_binary value.

SetBinary(ul_column_num, p_ul_binary) Method [page 131]
Sets a column to a ul_binary value.

1.7.31.1 SetBinary(const char *, p_ul_binary) Method

Sets a column to a ul_binary value.

Syntax

```
public virtual bool SetBinary (
    const char * cname,
    p_ul_binary value
)
```

Parameters

cname The name of the column.
value The ul_binary value. Passing NULL is equivalent to calling the SetNull method.

Returns

True on success; otherwise, returns false.

1.7.31.2 SetBinary(ul_column_num, p_ul_binary) Method

Sets a column to a ul_binary value.

Syntax

```
public virtual bool SetBinary (
    ul_column_num cid,
    p_ul_binary value
)
```

Parameters

cid The 1-based ordinal column number.

value The ul_binary value. Passing NULL is equivalent to calling the SetNull method.

Returns

True on success; otherwise, returns false.

1.7.32 SetBytes(ul_column_num cid, const ul_byte * value, size_t len) Method

Syntax

```
public virtual size_t SetBytes (
    ul_column_num cid,
    ul_byte * dst,
    size_t len
)
```

Parameters

cid The 1-based ordinal column number.
dst The byte array value. Passing NULL is equivalent to calling the SetNull method.
len The size of the array value.

Returns

True on success; otherwise, returns false.

1.7.33 SetIntWithType Method

Sets a column to an integer value of the specified integer type.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetIntWithType(const char *, ul_s_big, ul_column_storage_type) [page 133]	Sets a column to an integer value of the specified integer type.
public virtual bool	SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) [page 134]	Sets a column to an integer value of the specified integer type.

In this section:

SetIntWithType(const char *, ul_s_big, ul_column_storage_type) Method [page 133]
Sets a column to an integer value of the specified integer type.

SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) Method [page 134]
Sets a column to an integer value of the specified integer type.

1.7.33.1 SetIntWithType(const char *, ul_s_big, ul_column_storage_type) Method

Sets a column to an integer value of the specified integer type.

Syntax

```
public virtual bool SetIntWithType (
    const char * cname,
    ul_s_big value,
    ul_column_storage_type type
)
```

Parameters

cname The name of the column.
value The integer value.
type The integer type to treat the value as.

Returns

True on success; otherwise, returns false.

Remarks

The following is a list of integer values that can be used for the value parameter:

- UL_TYPE_BIT
- UL_TYPE_TINY
- UL_TYPE_S_SHORT
- UL_TYPE_U_SHORT
- UL_TYPE_S_LONG
- UL_TYPE_U_LONG
- UL_TYPE_S_BIG
- UL_TYPE_U_BIG

1.7.33.2 SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) Method

Sets a column to an integer value of the specified integer type.

Syntax

```
public virtual bool SetIntWithType (
    ul_column_num cid,
    ul_s_big value,
    ul_column_storage_type type
)
```

Parameters

cid The 1-based ordinal column number.
value The integer value.
type The integer type to treat the value as.

Returns

True on success; otherwise, returns false.

Remarks

The following is a list of integer values that can be used for the value parameter:

- UL_TYPE_BIT
- UL_TYPE_TINY
- UL_TYPE_S_SHORT
- UL_TYPE_U_SHORT
- UL_TYPE_S_LONG
- UL_TYPE_U_LONG
- UL_TYPE_S_BIG
- UL_TYPE_U_BIG

1.7.34 SetDateTime Method

Sets a column to a DECL_DATETIME value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetDateTime(const char *, DECL_DATETIME *) [page 135]	Sets a column to a DECL_DATETIME value.
public virtual bool	SetDateTime(ul_column_num, DECL_DATETIME *) [page 136]	Sets a column to a DECL_DATETIME value.

In this section:

SetDateTime(const char *, DECL_DATETIME *) Method [page 135]
Sets a column to a DECL_DATETIME value.

SetDateTime(ul_column_num, DECL_DATETIME *) Method [page 136]
Sets a column to a DECL_DATETIME value.

1.7.34.1 SetDateTime(const char *, DECL_DATETIME *) Method

Sets a column to a DECL_DATETIME value.

Syntax

```
public virtual bool SetDateTime (
```

```
    const char * cname,
    DECL_DATETIME * value
)
```

Parameters

cname The name of the column.
value The DECL_DATETIME value. Passing NULL is equivalent to calling the SetNull method.

Returns

True on success; otherwise, returns false.

1.7.34.2 SetDateTime(ul_column_num, DECL_DATETIME *) Method

Sets a column to a DECL_DATETIME value.

Syntax

```
public virtual bool SetDateTime (
    ul_column_num cid,
    DECL_DATETIME * value
)
```

Parameters

cid The 1-based ordinal column number.
value The DECL_DATETIME value. Passing NULL is equivalent to calling the SetNull method.

Returns

True on success; otherwise, returns false.

1.7.35 SetDefault Method

Sets a column to its default value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetDefault(const char *) [page 137]	Sets a column to its default value.
public virtual bool	SetDefault(ul_column_num) [page 138]	Sets a column to its default value.

In this section:

SetDefault(const char *) Method [page 137]
Sets a column to its default value.

SetDefault(ul_column_num) Method [page 138]
Sets a column to its default value.

1.7.35.1 SetDefault(const char *) Method

Sets a column to its default value.

Syntax

```
public virtual bool SetDefault (const char * cname)
```

Parameters

cname The name of the column.

Returns

True on success; otherwise, returns false.

1.7.35.2 SetDefault(ul_column_num) Method

Sets a column to its default value.

Syntax

```
public virtual bool SetDefault (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

True on success; otherwise, returns false.

1.7.36 SetDouble Method

Sets a column to a double value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetDouble(const char *, ul_double) [page 139]	Sets a column to a double value.
public virtual bool	SetDouble(ul_column_num, ul_double) [page 139]	Sets a column to a double value.

In this section:

SetDouble(const char *, ul_double) Method [page 139]
Sets a column to a double value.

SetDouble(ul_column_num, ul_double) Method [page 139]
Sets a column to a double value.

1.7.36.1 SetDouble(const char *, ul_double) Method

Sets a column to a double value.

Syntax

```
public virtual bool SetDouble (
    const char * cname,
    ul_double value
)
```

Parameters

cname The name of the column.
value The double value.

Returns

True on success; otherwise, returns false.

1.7.36.2 SetDouble(ul_column_num, ul_double) Method

Sets a column to a double value.

Syntax

```
public virtual bool SetDouble (
    ul_column_num cid,
    ul_double value
)
```

Parameters

cid The 1-based ordinal column number.
value The double value.

Returns

True on success; otherwise, returns false.

1.7.37 SetFloat Method

Sets a column to a float value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetFloat(const char *, ul_real) [page 140]	Sets a column to a float value.
public virtual bool	SetFloat(ul_column_num, ul_real) [page 141]	Sets a column to a float value.

In this section:

SetFloat(const char *, ul_real) Method [page 140]
Sets a column to a float value.

SetFloat(ul_column_num, ul_real) Method [page 141]
Sets a column to a float value.

1.7.37.1 SetFloat(const char *, ul_real) Method

Sets a column to a float value.

Syntax

```
public virtual bool SetFloat (
    const char * cname,
    ul_real value
)
```

Parameters

cname The name of the column.
value The float value.

Returns

True on success; otherwise, returns false.

1.7.37.2 SetFloat(ul_column_num, ul_real) Method

Sets a column to a float value.

Syntax

```
public virtual bool SetFloat (
    ul_column_num cid,
    ul_real value
)
```

Parameters

cid The 1-based ordinal column number.
value The float value.

Returns

True on success; otherwise, returns false.

1.7.38 SetGuid Method

Sets a column to a GUID value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetGuid(const char *, GUID *) [page 142]	Sets a column to a GUID value.
public virtual bool	SetGuid(ul_column_num, GUID *) [page 142]	Sets a column to a GUID value.

In this section:

1.7.38.1 SetGuid(const char *, GUID *) Method

Sets a column to a GUID value.

Syntax

```
public virtual bool SetGuid (
    const char * cname,
    GUID * value
)
```

Parameters

cname The name of the column.
value The GUID value. Passing NULL is equivalent to calling the SetNull method.

Returns

True on success; otherwise, returns false.

1.7.38.2 SetGuid(ul_column_num, GUID *) Method

Sets a column to a GUID value.

Syntax

```
public virtual bool SetGuid (
    ul_column_num cid,
    GUID * value
)
```

Parameters

cid The 1-based ordinal column number.
value The GUID value. Passing NULL is equivalent to calling the SetNull method.

Returns

True on success; otherwise, returns false.

1.7.39 SetInt Method

Sets a column to an integer value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetInt(const char *, ul_s_long) [page 143]	Sets a column to an integer value.
public virtual bool	SetInt(ul_column_num, ul_s_long) [page 144]	Sets a column to an integer value.

In this section:

SetInt(const char *, ul_s_long) Method [page 143]
Sets a column to an integer value.

SetInt(ul_column_num, ul_s_long) Method [page 144]
Sets a column to an integer value.

1.7.39.1 SetInt(const char *, ul_s_long) Method

Sets a column to an integer value.

Syntax

```
public virtual bool SetInt (
    const char * cname,
    ul_s_long value
)
```

Parameters

cname The name of the column.
value The signed integer value.

Returns

True on success; otherwise, returns false.

1.7.39.2 SetInt(ul_column_num, ul_s_long) Method

Sets a column to an integer value.

Syntax

```
public virtual bool SetInt (
    ul_column_num cid,
    ul_s_long value
)
```

Parameters

cid The 1-based ordinal column number.
value The signed integer value.

Returns

True on success; otherwise, returns false.

1.7.40 SetNull Method

Sets a column to null.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetNull(const char *) [page 145]	Sets a column to null.
public virtual bool	SetNull(ul_column_num) [page 146]	Sets a column to null.

In this section:

SetNull(const char *) Method [page 145]
Sets a column to null.

SetNull(ul_column_num) Method [page 146]
Sets a column to null.

1.7.40.1 SetNull(const char *) Method

Sets a column to null.

Syntax

```
public virtual bool SetNull (const char * cname)
```

Parameters

cname The name of the column.

Returns

True on success; otherwise, returns false.

1.7.40.2 SetNull(ul_column_num) Method

Sets a column to null.

Syntax

```
public virtual bool SetNull (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

True on success; otherwise, returns false.

1.7.41 SetString Method

Sets a column to a string value.

Overload list

Modifier and Type	Overload name	Description
public virtual bool	SetString(const char *, const char *, size_t) [page 147]	Sets a column to a string value.
public virtual bool	SetString(ul_column_num, const char *, size_t) [page 147]	Sets a column to a string value.

In this section:

SetString(const char *, const char *, size_t) Method [page 147]
Sets a column to a string value.

SetString(ul_column_num, const char *, size_t) Method [page 147]
Sets a column to a string value.

1.7.41.1 SetString(const char *, const char *, size_t) Method

Sets a column to a string value.

Syntax

```
public virtual bool SetString (
    const char * cname,
    const char * value,
    size_t len
)
```

Parameters

cname The name of the column.
value The string value. Passing NULL is equivalent to calling the SetNull method.
len Optional. The length of the string in bytes or the UL_NULL_TERMINATED_STRING constant if the string is null-terminated. The SQLE_INVALID_PARAMETER constant is set if the len value is set larger than 32K. For large strings, call the AppendStringChunk method instead.

Returns

True on success; otherwise, returns false.

Related Information

AppendStringChunk(ul_column_num, const char *, size_t) Method [page 99]

1.7.41.2 SetString(ul_column_num, const char *, size_t) Method

Sets a column to a string value.

Syntax

```
public virtual bool SetString (
    ul_column_num cid,
    const char * value,
    size_t len
)
```

Parameters

cid The 1-based ordinal column number.
value The string value. Passing NULL is equivalent to calling the SetNull method.
len Optional. The length of the string in bytes or the UL_NULL_TERMINATED_STRING constant if the string is null-terminated. The SQLE_INVALID_PARAMETER constant is set if the len value is set larger than 32K. For large strings, call the AppendStringChunk method instead.

Returns

True on success; otherwise, returns false.

Related Information

AppendStringChunk(ul_column_num, const char *, size_t) Method [page 99]

1.7.42 Update() Method

Updates the current row.

Syntax

```
public virtual bool Update ()
```

Returns

True on success, otherwise false.

1.7.43 UpdateBegin() Method

Selects the update mode for setting columns.

Syntax

```
public virtual bool UpdateBegin ()
```

Returns

True on success, otherwise false.

Remarks

Columns in the primary key may not be modified when in update mode.

1.8 ULResultSetSchema Class

Represents the schema of an UltraLite result set.

Syntax

```
public class ULResultSetSchema
```

Members

All members of ULResultSetSchema, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual ul_column_num	GetColumnCount() [page 150]	Gets the number of columns in the result set or table.
public virtual ul_column_num	GetColumnID(const char *) [page 151]	Gets the 1-based column ID from its name.
public virtual const char *	GetColumnName(ul_column_num, ul_column_name_type) [page 151]	Gets the name of a column given its 1-based ID.
public virtual size_t	GetColumnPrecision(ul_column_num) [page 152]	Gets the precision of a numeric column.
public virtual size_t	GetColumnScale(ul_column_num) [page 153]	Gets the scale of a numeric column.
public virtual size_t	GetColumnSize(ul_column_num) [page 153]	Gets the size of the column.
public virtual ul_column_sql_type	GetColumnSQLType(ul_column_num) [page 154]	Gets the SQL type of a column.

Modifier and Type	Method	Description
public virtual ul_column_storage_type	GetColumnType(ul_column_num) [page 154]	Gets the storage/host variable type of a column.
public virtual ULConnection *	GetConnection() [page 155]	Gets the ULConnection object.
public virtual bool	IsAliased(ul_column_num) [page 155]	Indicates whether the column in a result set was given an alias.

In this section:

GetColumnCount() Method [page 150]
Gets the number of columns in the result set or table.

GetColumnID(const char *) Method [page 151]
Gets the 1-based column ID from its name.

GetColumnName(ul_column_num, ul_column_name_type) Method [page 151]
Gets the name of a column given its 1-based ID.

GetColumnPrecision(ul_column_num) Method [page 152]
Gets the precision of a numeric column.

GetColumnScale(ul_column_num) Method [page 153]
Gets the scale of a numeric column.

GetColumnSize(ul_column_num) Method [page 153]
Gets the size of the column.

GetColumnSQLType(ul_column_num) Method [page 154]
Gets the SQL type of a column.

GetColumnType(ul_column_num) Method [page 154]
Gets the storage/host variable type of a column.

GetConnection() Method [page 155]
Gets the ULConnection object.

IsAliased(ul_column_num) Method [page 155]
Indicates whether the column in a result set was given an alias.

1.8.1 GetColumnCount() Method

Gets the number of columns in the result set or table.

Syntax

```
public virtual ul_column_num GetColumnCount ()
```

Returns

The number of columns in the result set or table.

1.8.2 GetColumnID(const char *) Method

Gets the 1-based column ID from its name.

Syntax

```
public virtual ul_column_num GetColumnID (const char * columnName)
```

Parameters

columnName The column name.

Returns

0 if the column does not exist; otherwise, returns SQLE_COLUMN_NOT_FOUND if the column name does not exist.

1.8.3 GetColumnName(ul_column_num, ul_column_name_type) Method

Gets the name of a column given its 1-based ID.

Syntax

```
public virtual const char * GetColumnName (
    ul_column_num cid,
    ul_column_name_type type
)
```

Parameters

cid The 1-based ordinal column number.

type The desired column name type.

Returns

A pointer to a string buffer containing the column name, if found. The pointer points to a static buffer whose contents may be changed by any subsequent UltraLite call, so you need to make a copy of the value if you need to keep it for a while. If the column does not exist, NULL is returned and SQLE_COLUMN_NOT_FOUND is set.

Remarks

Depending on the type selected and how the column was declared in the SELECT statement, the column name may be returned in the form [table- name].[column-name].

The type parameter is used to specify what type of column name to return.

Related Information

ul_column_name_type Enumeration [page 179]

1.8.4 GetColumnPrecision(ul_column_num) Method

Gets the precision of a numeric column.

Syntax

```
public virtual size_t GetColumnPrecision (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

0 if the column is not a numeric type or if the column does not exist. SQLE_COLUMN_NOT_FOUND is set if the column name does not exist. SQLE_DATATYPE_NOT_ALLOWED is set if the column type is not numeric.

1.8.5 GetColumnScale(ul_column_num) Method

Gets the scale of a numeric column.

Syntax

```
public virtual size_t GetColumnScale (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

0 if the column is not a numeric type or if the column does not exist. SQLE_COLUMN_NOT_FOUND is set if the column name does not exist. SQLE_DATATYPE_NOT_ALLOWED is set if the column type is not numeric.

1.8.6 GetColumnSize(ul_column_num) Method

Gets the size of the column.

Syntax

```
public virtual size_t GetColumnSize (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

0 if the column does not exist or if the column type does not have a variable length. SQLE_COLUMN_NOT_FOUND is set if the column name does not exist. SQLE_DATATYPE_NOT_ALLOWED is set if the column type is not UL_SQLTYPE_CHAR or UL_SQLTYPE_BINARY.

1.8.7 GetColumnSQLType(ul_column_num) Method

Gets the SQL type of a column.

Syntax

```
public virtual ul_column_sql_type GetColumnSQLType (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

UL_SQLTYPE_BAD_INDEX if the column does not exist.

1.8.8 GetColumnType(ul_column_num) Method

Gets the storage/host variable type of a column.

Syntax

```
public virtual ul_column_storage_type GetColumnType (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

UL_TYPE_BAD_INDEX if the column does not exist.

1.8.9 GetConnection() Method

Gets the ULConnection object.

Syntax

```
public virtual ULConnection * GetConnection ()
```

Returns

The ULConnection object associated with this result set schema.

1.8.10 IsAliased(ul_column_num) Method

Indicates whether the column in a result set was given an alias.

Syntax

```
public virtual bool IsAliased (ul_column_num cid)
```

Parameters

cid The 1-based ordinal column number.

Returns

True if the column is aliased; otherwise, returns false.

1.9 ULTable Class

Represents a table in an UltraLite database.

Syntax

```
public class ULTable : ULResultSet
```

Members

All members of ULTable, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual bool	DeleteAllRows() [page 160]	Deletes all rows from a table.
public virtual bool	Find(ul_column_num) [page 161]	Performs an exact match lookup based on the current index scanning forward through the table.
public virtual bool	FindBegin() [page 162]	Prepares to perform a new Find call on a table by entering find mode.
public virtual bool	FindFirst(ul_column_num) [page 162]	Performs an exact match lookup based on the current index scanning forward through the table.
public virtual bool	FindLast(ul_column_num) [page 163]	Performs an exact match lookup based on the current index scanning backward through the table.
public virtual bool	FindNext(ul_column_num) [page 163]	Gets the next row that exactly matches the index.
public virtual bool	FindPrevious(ul_column_num) [page 164]	Gets the previous row that exactly matches the index.
public virtual ULTableSchema *	GetTableSchema() [page 164]	Returns a ULTableSchema object that can be used to get schema information about the table.
public virtual bool	Insert() [page 165]	Inserts a new row into the table.
public virtual bool	InsertBegin() [page 165]	Selects the insert mode for setting columns.
public virtual bool	Lookup(ul_column_num) [page 166]	Performs a lookup based on the current index scanning forward through the table.
public virtual bool	LookupBackward(ul_column_num) [page 166]	Performs a lookup based on the current index scanning backward through the table.
public virtual bool	LookupBegin() [page 167]	Prepares to perform a new lookup on a table.
public virtual bool	LookupForward(ul_column_num) [page 167]	Performs a lookup based on the current index scanning forward through the table.
public virtual bool	TruncateTable() [page 168]	Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.

Inherited members from ULResultSet

Modifier and Type	Member	Description
public virtual bool	AfterLast() [page 95]	Moves the cursor after the last row.
public virtual bool	AppendByteChunk(ul_column_num, const ul_byte *, size_t) [page 97]	Appends bytes to a column.
public virtual bool	AppendByteChunk(const char *, const ul_byte *, size_t) [page 96]	Appends bytes to a column.
public virtual bool	AppendStringChunk(ul_column_num, const char *, size_t) [page 99]	Appends a string chunk to a column.
public virtual bool	AppendStringChunk(const char *, const char *, size_t) [page 98]	Appends a string chunk to a column.
public virtual bool	BeforeFirst() [page 100]	Moves the cursor before the first row.
public virtual void	Close() [page 100]	Destroys this object.
public virtual bool	Delete() [page 101]	Deletes the current row and moves it to the next valid row.
public virtual bool	DeleteNamed(const char *) [page 101]	Deletes the current row and moves it to the next valid row.
public virtual bool	First() [page 101]	Moves the cursor to the first row.
public virtual bool	GetBinary(ul_column_num, p_ul_binary, size_t) [page 103]	Fetches a value from a column as a ul_binary value.
public virtual bool	GetBinary(const char *, p_ul_binary, size_t) [page 102]	Fetches a value from a column as a ul_binary value.
public virtual size_t	GetBinaryLength(ul_column_num) [page 105]	Gets the binary length of the value of a column.
public virtual size_t	GetBinaryLength(const char *) [page 104]	Gets the binary length of the value of a column.
public virtual size_t	GetByteChunk(ul_column_num, ul_byte *, size_t, size_t) [page 107]	Gets a binary chunk from the column.
public virtual size_t	GetByteChunk(const char *, ul_byte *, size_t, size_t) [page 106]	Gets a binary chunk from the column.
public virtual ULConnection *	GetConnection() [page 108]	Gets the connection object.
public virtual bool	GetDateTime(ul_column_num, DECL_DATETIME *) [page 110]	Fetches a value from a column as a DECL_DATETIME.
public virtual bool	GetDateTime(const char *, DECL_DATETIME *) [page 109]	Fetches a value from a column as a DECL_DATETIME.
public virtual ul_double	GetDouble(ul_column_num) [page 111]	Fetches a value from a column as a double.
public virtual ul_double	GetDouble(const char *) [page 111]	Fetches a value from a column as a double.
public virtual ul_real	GetFloat(ul_column_num) [page 113]	Fetches a value from a column as a float.
public virtual ul_real	GetFloat(const char *) [page 112]	Fetches a value from a column as a float.

Modifier and Type	Member	Description
public virtual bool	GetGuid(ul_column_num, GUID *) [page 115]	Fetches a value from a column as a GUID.
public virtual bool	GetGuid(const char *, GUID *) [page 114]	Fetches a value from a column as a GUID.
public virtual ul_s_long	GetInt(ul_column_num) [page 116]	Fetches a value from a column as an integer.
public virtual ul_s_long	GetInt(const char *) [page 116]	Fetches a value from a column as an integer.
public virtual ul_s_big	GetIntWithType(ul_column_num, ul_column_storage_type) [page 118]	Fetches a value from a column as the specified integer type.
public virtual ul_s_big	GetIntWithType(const char *, ul_column_storage_type) [page 117]	Fetches a value from a column as the specified integer type.
public virtual const ULResultSetSchema &	GetResultSetSchema() [page 119]	Returns an object that can be used to get information about the result set.
public virtual ul_u_long	GetRowCount(ul_u_long) [page 119]	Gets the number of rows in the table.
public virtual UL_RS_STATE	GetState() [page 120]	Gets the internal state of the cursor.
public virtual bool	GetString(ul_column_num, char *, size_t) [page 121]	Fetches a value from a column as a null-terminated string.
public virtual bool	GetString(const char *, char *, size_t) [page 121]	Fetches a value from a column as a null-terminated string.
public virtual size_t	GetStringChunk(ul_column_num, char *, size_t, size_t) [page 124]	Gets a string chunk from the column.
public virtual size_t	GetStringChunk(const char *, char *, size_t, size_t) [page 123]	Gets a string chunk from the column.
public virtual size_t	GetStringLength(ul_column_num) [page 126]	Gets the string length of the value of a column.
public virtual size_t	GetStringLength(const char *) [page 125]	Gets the string length of the value of a column.
public virtual bool	IsNull(ul_column_num) [page 128]	Checks if a column is NULL.
public virtual bool	IsNull(const char *) [page 128]	Checks if a column is NULL.
public virtual bool	Last() [page 129]	Moves the cursor to the last row.
public virtual bool	Next() [page 129]	Moves the cursor forward one row.
public virtual bool	Previous() [page 129]	Moves the cursor back one row.
public virtual bool	Relative(ul_fetch_offset) [page 130]	Moves the cursor by offset rows from the current cursor position.
public virtual bool	SetBinary(ul_column_num, p_ul_binary) [page 131]	Sets a column to a ul_binary value.
public virtual bool	SetBinary(const char *, p_ul_binary) [page 131]	Sets a column to a ul_binary value.
public virtual bool	SetDateTime(ul_column_num, DECL_DATETIME *) [page 136]	Sets a column to a DECL_DATETIME value.

Modifier and Type	Member	Description
public virtual bool	SetDateTime(const char *, DECL_DATETIME *) [page 135]	Sets a column to a DECL_DATETIME value.
public virtual bool	SetDefault(ul_column_num) [page 138]	Sets a column to its default value.
public virtual bool	SetDefault(const char *) [page 137]	Sets a column to its default value.
public virtual bool	SetDouble(ul_column_num, ul_double) [page 139]	Sets a column to a double value.
public virtual bool	SetDouble(const char *, ul_double) [page 139]	Sets a column to a double value.
public virtual bool	SetFloat(ul_column_num, ul_real) [page 141]	Sets a column to a float value.
public virtual bool	SetFloat(const char *, ul_real) [page 140]	Sets a column to a float value.
public virtual bool	SetGuid(ul_column_num, GUID *) [page 142]	Sets a column to a GUID value.
public virtual bool	SetGuid(const char *, GUID *) [page 142]	Sets a column to a GUID value.
public virtual bool	SetInt(ul_column_num, ul_s_long) [page 144]	Sets a column to an integer value.
public virtual bool	SetInt(const char *, ul_s_long) [page 143]	Sets a column to an integer value.
public virtual bool	SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type) [page 134]	Sets a column to an integer value of the specified integer type.
public virtual bool	SetIntWithType(const char *, ul_s_big, ul_column_storage_type) [page 133]	Sets a column to an integer value of the specified integer type.
public virtual bool	SetNull(ul_column_num) [page 146]	Sets a column to null.
public virtual bool	SetNull(const char *) [page 145]	Sets a column to null.
public virtual bool	SetString(ul_column_num, const char *, size_t) [page 147]	Sets a column to a string value.
public virtual bool	SetString(const char *, const char *, size_t) [page 147]	Sets a column to a string value.
public virtual bool	Update() [page 148]	Updates the current row.
public virtual bool	UpdateBegin() [page 148]	Selects the update mode for setting columns.

In this section:

DeleteAllRows() Method [page 160]
Deletes all rows from a table.

Find(ul_column_num) Method [page 161]
Performs an exact match lookup based on the current index scanning forward through the table.

FindBegin() Method [page 162]

Prepares to perform a new Find call on a table by entering find mode.

FindFirst(ul_column_num) Method [page 162]
Performs an exact match lookup based on the current index scanning forward through the table.

FindLast(ul_column_num) Method [page 163]
Performs an exact match lookup based on the current index scanning backward through the table.

FindNext(ul_column_num) Method [page 163]
Gets the next row that exactly matches the index.

FindPrevious(ul_column_num) Method [page 164]
Gets the previous row that exactly matches the index.

GetTableSchema() Method [page 164]
Returns a ULTableSchema object that can be used to get schema information about the table.

Insert() Method [page 165]
Inserts a new row into the table.

InsertBegin() Method [page 165]
Selects the insert mode for setting columns.

Lookup(ul_column_num) Method [page 166]
Performs a lookup based on the current index scanning forward through the table.

LookupBackward(ul_column_num) Method [page 166]
Performs a lookup based on the current index scanning backward through the table.

LookupBegin() Method [page 167]
Prepares to perform a new lookup on a table.

LookupForward(ul_column_num) Method [page 167]
Performs a lookup based on the current index scanning forward through the table.

TruncateTable() Method [page 168]
Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.

1.9.1 DeleteAllRows() Method

Deletes all rows from a table.

Syntax

```
public virtual bool DeleteAllRows ()
```

Returns

True on success; otherwise, returns false. For example, false is returned when the table is not open, or a SQL error occurred.

Remarks

In some applications, you may want to delete all rows from a table before downloading a new set of data into the table. If you set the stop synchronization property on the connection, the deleted rows are not synchronized.

> **i** Note
>
> Any uncommitted inserts from other connections are not deleted. They are also not deleted if the other connection performs a rollback after it calls the DeleteAllRows method.

If this table has been opened without an index, then it is considered read-only and data cannot be deleted.

1.9.2 Find(ul_column_num) Method

Performs an exact match lookup based on the current index scanning forward through the table.

> Syntax

```
public virtual bool Find (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use during the search.

Returns

If no row matches the index value, the cursor position is set after the last row and the method returns false.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the first row that exactly matches the index value.

1.9.3 FindBegin() Method

Prepares to perform a new Find call on a table by entering find mode.

Syntax

```
public virtual bool FindBegin ()
```

Returns

True on success; otherwise, returns false.

Remarks

You may only set columns in the index that the table was opened with. This method cannot be called if the table was opened without an index.

1.9.4 FindFirst(ul_column_num) Method

Performs an exact match lookup based on the current index scanning forward through the table.

Syntax

```
public virtual bool FindFirst (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use during the search.

Returns

If no row matches the index value, the cursor position is set after the last row and the method returns false.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the first row that exactly matches the index value.

1.9.5 FindLast(ul_column_num) Method

Performs an exact match lookup based on the current index scanning backward through the table.

Syntax

```
public virtual bool FindLast (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use during the search.

Returns

If no row matches the index value, the cursor position is set before the first row and the method returns false.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the first row that exactly matches the index value.

1.9.6 FindNext(ul_column_num) Method

Gets the next row that exactly matches the index.

Syntax

```
public virtual bool FindNext (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use during the search.

Returns

False if no more rows match the index. In this case, the cursor is positioned after the last row.

1.9.7 FindPrevious(ul_column_num) Method

Gets the previous row that exactly matches the index.

Syntax

```
public virtual bool FindPrevious (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use during the search.

Returns

False if no more rows match the index. In this case, the cursor is positioned before the first row.

1.9.8 GetTableSchema() Method

Returns a ULTableSchema object that can be used to get schema information about the table.

Syntax

```
public virtual ULTableSchema * GetTableSchema ()
```

Returns

A ULTableSchema object that can be used to get schema information about the table.

1.9.9 Insert() Method

Inserts a new row into the table.

Syntax

```
public virtual bool Insert ()
```

Returns

True on success; otherwise returns false.

1.9.10 InsertBegin() Method

Selects the insert mode for setting columns.

Syntax

```
public virtual bool InsertBegin ()
```

Returns

True on success; otherwise, returns false.

Remarks

All columns are set to their default value during an insert unless an alternative value is supplied via Set method calls.

1.9.11 Lookup(ul_column_num) Method

Performs a lookup based on the current index scanning forward through the table.

Syntax

```
public virtual bool Lookup (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use in the lookup.

Returns

False if the resulting cursor position is set after the last row.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the last row that matches or is less than the index value. For composite indexes, the ncols parameter specifies the number of columns to use in the lookup.

1.9.12 LookupBackward(ul_column_num) Method

Performs a lookup based on the current index scanning backward through the table.

Syntax

```
public virtual bool LookupBackward (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use in the lookup.

Returns

False if the resulting cursor position is set before the first row.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the last row that matches or is less than the index value. For composite indexes, the ncols parameter specifies the number of columns to use in the lookup.

1.9.13 LookupBegin() Method

Prepares to perform a new lookup on a table.

Syntax

```
public virtual bool LookupBegin ()
```

Returns

True on success; otherwise, returns false.

Remarks

You may only set columns in the index that the table was opened with. If the table was opened without an index, this method cannot be called.

1.9.14 LookupForward(ul_column_num) Method

Performs a lookup based on the current index scanning forward through the table.

Syntax

```
public virtual bool LookupForward (ul_column_num ncols)
```

Parameters

ncols For composite indexes, the number of columns to use in the lookup.

Returns

False if the resulting cursor position is set after the last row.

Remarks

To specify the value to search for, set the column value for each column in the index. The cursor is positioned on the last row that matches or is less than the index value. For composite indexes, the ncols parameter specifies the number of columns to use in the lookup.

1.9.15 TruncateTable() Method

Truncates the table and temporarily activates STOP SYNCHRONIZATION DELETE.

Syntax

```
public virtual bool TruncateTable ()
```

Returns

True on success; otherwise, returns false.

1.10 ULTableSchema Class

Represents the schema of an UltraLite table.

Syntax

```
public class ULTableSchema : ULResultSetSchema
```

Members

All members of ULTableSchema, including inherited members.

Methods

Modifier and Type	Method	Description
public virtual void	Close() [page 171]	Destroys this object.
public virtual const char *	GetColumnDefault(ul_column_num) [page 171]	Gets the default value for the column if it exists.
public virtual ul_column_default_type	GetColumnDefaultType(ul_column_num) [page 171]	Gets the type of column default.
public virtual bool	GetGlobalAutoincPartitionSize(ul_column_num, ul_u_big *) [page 172]	Gets the partition size.
public virtual ul_index_num	GetIndexCount() [page 173]	Gets the number of indexes in the table.
public virtual ULIndexSchema *	GetIndexSchema(const char *) [page 173]	Gets the schema of an index, given the name.
public virtual const char *	GetName() [page 174]	Gets the name of the table.
public virtual ULIndexSchema *	GetNextIndex(ul_index_iter *) [page 174]	Gets the next index (schema) in the table.
public virtual const char *	GetOptimalIndex(ul_column_num) [page 175]	Determines the best index to use for searching for a column value.
public virtual ULIndexSchema *	GetPrimaryKey() [page 175]	Gets the primary key for the table.
public virtual const char *	GetPublicationPredicate(const char *) [page 176]	Gets the publication predicate as a string.
public virtual ul_table_sync_type	GetTableSyncType() [page 176]	Gets the table synchronization type.
public virtual bool	InPublication(const char *) [page 177]	Checks whether the table is contained in the named publication.
public virtual bool	IsColumnInIndex(ul_column_num, const char *) [page 177]	Checks whether the column is contained in the named index.
public virtual bool	IsColumnNullable(ul_column_num) [page 178]	Checks whether the specified column is nullable.

Inherited members from ULResultSetSchema

Modifier and Type	Member	Description
public virtual ul_column_num	GetColumnCount() [page 150]	Gets the number of columns in the result set or table.
public virtual ul_column_num	GetColumnID(const char *) [page 151]	Gets the 1-based column ID from its name.
public virtual const char *	GetColumnName(ul_column_num, ul_column_name_type) [page 151]	Gets the name of a column given its 1-based ID.
public virtual size_t	GetColumnPrecision(ul_column_num) [page 152]	Gets the precision of a numeric column.

Modifier and Type	Member	Description
public virtual size_t	GetColumnScale(ul_column_num) [page 153]	Gets the scale of a numeric column.
public virtual size_t	GetColumnSize(ul_column_num) [page 153]	Gets the size of the column.
public virtual ul_column_sql_type	GetColumnSQLType(ul_column_num) [page 154]	Gets the SQL type of a column.
public virtual ul_column_storage_type	GetColumnType(ul_column_num) [page 154]	Gets the storage/host variable type of a column.
public virtual ULConnection *	GetConnection() [page 155]	Gets the ULConnection object.
public virtual bool	IsAliased(ul_column_num) [page 155]	Indicates whether the column in a result set was given an alias.

In this section:

Close() Method [page 171]
Destroys this object.

GetColumnDefault(ul_column_num) Method [page 171]
Gets the default value for the column if it exists.

GetColumnDefaultType(ul_column_num) Method [page 171]
Gets the type of column default.

GetGlobalAutoincPartitionSize(ul_column_num, ul_u_big *) Method [page 172]
Gets the partition size.

GetIndexCount() Method [page 173]
Gets the number of indexes in the table.

GetIndexSchema(const char *) Method [page 173]
Gets the schema of an index, given the name.

GetName() Method [page 174]
Gets the name of the table.

GetNextIndex(ul_index_iter *) Method [page 174]
Gets the next index (schema) in the table.

GetOptimalIndex(ul_column_num) Method [page 175]
Determines the best index to use for searching for a column value.

GetPrimaryKey() Method [page 175]
Gets the primary key for the table.

GetPublicationPredicate(const char *) Method [page 176]
Gets the publication predicate as a string.

GetTableSyncType() Method [page 176]
Gets the table synchronization type.

InPublication(const char *) Method [page 177]
Checks whether the table is contained in the named publication.

IsColumnInIndex(ul_column_num, const char *) Method [page 177]

Checks whether the column is contained in the named index.

IsColumnNullable(ul_column_num) Method [page 178]
Checks whether the specified column is nullable.

1.10.1 Close() Method

Destroys this object.

Syntax

```
public virtual void Close ()
```

1.10.2 GetColumnDefault(ul_column_num) Method

Gets the default value for the column if it exists.

Syntax

```
public virtual const char * GetColumnDefault (ul_column_num cid)
```

Parameters

cid A 1-based ordinal column number.

Returns

The default value. An empty string is returned if the column has no default value. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

1.10.3 GetColumnDefaultType(ul_column_num) Method

Gets the type of column default.

Syntax

```
public virtual ul_column_default_type GetColumnDefaultType (ul_column_num cid)
```

Parameters

cid A 1-based ordinal column number.

Returns

The type of column default.

Related Information

ul_column_default_type Enumeration [page 178]

1.10.4 GetGlobalAutoincPartitionSize(ul_column_num, ul_u_big *) Method

Gets the partition size.

Syntax

```
public virtual bool GetGlobalAutoincPartitionSize (
    ul_column_num cid,
    ul_u_big * size
)
```

Parameters

cid A 1-based ordinal column number.

size An output parameter. The partition size for the column. All global autoincrement columns in a given table share the same global autoincrement partition.

Returns

True on success; otherwise, returns false.

1.10.5 GetIndexCount() Method

Gets the number of indexes in the table.

Syntax

```
public virtual ul_index_num GetIndexCount ()
```

Returns

The number of indexes in the table.

Remarks

Index IDs and counts may change during a schema upgrade. To correctly identify an index, access it by name or refresh any cached IDs and counts after a schema upgrade.

1.10.6 GetIndexSchema(const char *) Method

Gets the schema of an index, given the name.

Syntax

```
public virtual ULIndexSchema * GetIndexSchema (const char * indexName)
```

Parameters

indexName The name of the index.

Returns

A ULIndexSchema object for the specified index, or NULL if the object does not exist.

1.10.7 GetName() Method

Gets the name of the table.

Syntax

```
public virtual const char * GetName ()
```

Returns

The name of the table. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

1.10.8 GetNextIndex(ul_index_iter *) Method

Gets the next index (schema) in the table.

Syntax

```
public virtual ULIndexSchema * GetNextIndex (ul_index_iter * iter)
```

Parameters

iter A pointer to the iterator variable.

Returns

A ULIndexSchema object, or NULL when the iteration is complete.

Remarks

Initialize the iter value to the ul_index_iter_start constant before the first call.

Related Information

ul_index_iter_start Variable [page 183]

1.10.9 GetOptimalIndex(ul_column_num) Method

Determines the best index to use for searching for a column value.

Syntax

```
public virtual const char * GetOptimalIndex (ul_column_num cid)
```

Parameters

cid A 1-based ordinal column number.

Returns

The name of the index or NULL if the column isn't indexed. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to keep it for a while.

1.10.10 GetPrimaryKey() Method

Gets the primary key for the table.

Syntax

```
public virtual ULIndexSchema * GetPrimaryKey ()
```

Returns

a ULIndexSchema object for the table's primary key.

1.10.11 GetPublicationPredicate(const char *) Method

Gets the publication predicate as a string.

Syntax

```
public virtual const char * GetPublicationPredicate (const char * pubName)
```

Parameters

pubName The name of the publication.

Returns

The publication predicate string for the specified publication. This value points to a static buffer whose contents may be changed by any subsequent UltraLite call, so make a copy of the value if you need to retain it.

1.10.12 GetTableSyncType() Method

Gets the table synchronization type.

Syntax

```
public virtual ul_table_sync_type GetTableSyncType ()
```

Returns

The table synchronization type.

Remarks

This method indicates how the table participates in synchronization, and is defined when the table is created with the SYNCHRONIZE constraint clause of the CREATE TABLE statement.

Related Information

ul_table_sync_type Enumeration [page 181]

1.10.13 InPublication(const char *) Method

Checks whether the table is contained in the named publication.

Syntax

```
public virtual bool InPublication (const char * pubName)
```

Parameters

pubName The name of the publication.

Returns

True if the table is contained in the publication; otherwise, returns false.

1.10.14 IsColumnInIndex(ul_column_num, const char *) Method

Checks whether the column is contained in the named index.

Syntax

```
public virtual bool IsColumnInIndex (
    ul_column_num cid,
    const char * indexName
)
```

Parameters

cid A 1-based ordinal column number.
indexName The name of the index.

Returns

True if the column is contained in the index; otherwise, returns false.

1.10.15 IsColumnNullable(ul_column_num) Method

Checks whether the specified column is nullable.

Syntax

```
public virtual bool IsColumnNullable (ul_column_num cid)
```

Parameters

cid A 1-based ordinal column number.

Returns

True if the column is nullable; otherwise, returns false.

1.11 ul_column_default_type Enumeration

Identifies a column default type.

Syntax

```
enum ul_column_default_type
```

Members

Member name	Description
ul_column_default_none	The column has no default value.
ul_column_default_autoincrement	The column default is AUTOINCREMENT.

Member name	Description
ul_column_default_global_autoincrement	The column default is GLOBAL AUTOINCREMENT.
ul_column_default_current_timestamp	The column default is CURRENT TIMESTAMP.
ul_column_default_current_utc_timestamp	The column default is CURRENT UTC TIMESTAMP.
ul_column_default_current_time	The column default is CURRENT TIME.
ul_column_default_current_date	The column default is CURRENT DATE.
ul_column_default_newid	The column default is NEWID().
ul_column_default_other	The column default is a user-specified constant.

Related Information

GetColumnDefaultType(ul_column_num) Method [page 171]

1.12 ul_column_name_type Enumeration

Specifies values that control how a column name is retrieved when describing a result set.

Syntax

```
enum ul_column_name_type
```

Members

Member name	Description
ul_name_type_sql	For SELECT statements, returns the alias or correlation name. For tables, returns the column name.
ul_name_type_sql_column_only	For SELECT statements, returns the alias or correlation name and exclude any table names that were specified. For tables, returns the column name.
ul_name_type_base_table	Returns the underlying table name if it can be determined. If the table does not exist in the database schema, returns an empty string.

Member name	Description
ul_name_type_base_column	Returns the underlying column name if it can be determined. If the column does not exist in the database schema, returns an empty string.
ul_name_type_qualified	Returns the underlying qualified column name, if it can be determined, when used in conjunction with the ULResultSetSchema.GetColumnName method. The returned name can be one of the following values, and is determined in this order: 1. The represented correlated table 2. The name of the represented table column 3. The alias name of the column 4. An empty string
ul_name_type_base	Indicates that a column name qualified with its table name should be returned when used with the GetColumnName method. If the column name being retrieved is associated with a base table in the query, then the base table name is used as the column qualifier (that is, the base_table_name.column_name value is returned). If the column name being retrieved refers to a column in a correlated table in the query, then the correlation name is used as the column qualifier (that is, the correl_table_name.col_name value is returned). If the column has an alias, then the qualified name of the column being aliased is returned; the alias is not part of the qualified name. Otherwise, an empty string is returned.

Related Information

GetColumnName(ul_column_num, ul_column_name_type) Method [page 151]

1.13 ul_index_flag Enumeration

Flags (bit fields) which identify properties of an index.

Syntax

```
enum ul_index_flag
```

Members

Member name	Description	Value
ul_index_flag_primary_key	The index is a primary key.	0x0001
ul_index_flag_unique_key	The index is a primary key or index created for a unique constraint (nulls not allowed).	0x0002
ul_index_flag_unique_index	The index was created with the UNIQUE flag (or is a primary key).	0x0004
ul_index_flag_foreign_key	The index is a foreign key.	0x0010
ul_index_flag_foreign_key_nullable	The foreign key allows nulls.	0x0020
ul_index_flag_foreign_key_check_on_commit	Referential integrity checks are performed on commit (rather than on insert/update).	0x0040

Related Information

GetIndexFlags() Method [page 74]

1.14 ul_table_sync_type Enumeration

Identifies a table synchronization type.

Syntax

```
enum ul_table_sync_type
```

Members

Member name	Description
ul_table_sync_on	All changed rows are synchronized, which is the default behavior. This initializer corresponds to the SYNCHRONIZE ON clause in a CREATE TABLE statement.

Member name	Description
ul_table_sync_off	Table is never synchronized. This initializer corresponds to the SYNCHRONIZE OFF clause in a CREATE TABLE statement.
ul_table_sync_upload_all_rows	Always upload every row, including unchanged rows. This initializer corresponds to the SYNCHRONIZE ALL clause in a CREATE TABLE statement.
ul_table_sync_download_only	Changes are never uploaded. This initializer corresponds to the SYNCHRONIZE DOWNLOAD clause in a CREATE TABLE statement.

Related Information

GetTableSyncType() Method [page 176]

1.15 UL_BLOB_CONTINUE Variable

Used when reading data with the ULResultSet.GetStringChunk or ULResultSet.GetByteChunk methods.

Syntax

```
#define UL_BLOB_CONTINUE
```

Remarks

This value indicates that the chunk of data to be read should continue from where the last chunk was read.

Related Information

GetStringChunk(ul_column_num, char *, size_t, size_t) Method [page 124]
GetByteChunk(ul_column_num, ul_byte *, size_t, size_t) Method [page 107]

1.16 ul_index_iter_start Variable

Used by the GetNextIndex method to initialize index iteration in a table.

Syntax

```
#define ul_index_iter_start
```

Related Information

GetNextIndex(ul_index_iter *) Method [page 174]

1.17 ul_publication_iter_start Variable

Used by the GetNextPublication method to initialize publication iteration in a database.

Syntax

```
#define ul_publication_iter_start
```

Related Information

GetNextPublication(ul_publication_iter *) Method [page 61]

1.18 ul_table_iter_start Variable

Used by the GetNextTable method to initialize table iteration in a database.

Syntax

```
#define ul_table_iter_start
```

Related Information

GetNextTable(ul_table_iter *) Method [page 62]

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP