PUBLIC

SQL Anywhere - UltraLite

Document Version: 17.01.0 – 2021-10-15

UltraLite - Embedded SQL API Reference

THE BEST RUN SAP

Content

1 UltraLite Embedded SQL API Reference. 5

1.1 db_fini Method. 9

1.2 db_init Method. 10

1.3 MLFileDownload(ml_file_transfer_info *) Method. 10

1.4 MLFileUpload(ml_file_transfer_info *) Method. 12

1.5 MLFiniFileTransferInfo(ml_file_transfer_info *) Method. 13

1.6 MLFTEnableRsaE2ee(ml_file_transfer_info *) Method. 13

1.7 MLFTEnableRsaEncryption(ml_file_transfer_info *) Method. 14

1.8 MLFTEnableRsaFipsE2ee(ml_file_transfer_info *) Method. 14

1.9 MLFTEnableRsaFipsEncryption(ml_file_transfer_info *) Method. 15

1.10 MLFTEnableZlibCompression(ml_file_transfer_info *) Method. 15

1.11 MLInitFileTransferInfo(ml_file_transfer_info *) Method. 16

1.12 ULCancelGetNotification(SQLCA *, char const *) Method. 17

1.13 ULChangeEncryptionKey(SQLCA *, char const *) Method. 17

1.14 ULCheckpoint(SQLCA *) Method. 18

1.15 ULCountUploadRows(SQLCA *, char const *, ul_u_long) Method. 18

1.16 ULCreateDatabase(SQLCA *, char const *, char const *, void *) Method. 20

1.17 ULCreateNotificationQueue(SQLCA *, char const *, char const *) Method. 21

1.18 ULDeclareEvent(SQLCA *, char const *) Method. 22

1.19 ULDeleteAllRows(SQLCA *, ul_table_num) Method. 23

1.20 ULDestroyNotificationQueue(SQLCA *, char const *) Method. 23

1.21 ULEnableAesDBEncryption(SQLCA *) Method. 24

1.22 ULEnableAesFipsDBEncryption(SQLCA *) Method. 25

1.23 ULEnableHttpSynchronization(SQLCA *) Method. 25

1.24 ULEnableRsaE2ee(SQLCA *) Method. 26

1.25 ULEnableRsaFipsE2ee(SQLCA *) Method. 26

1.26 ULEnableRsaFipsSyncEncryption(SQLCA *) Method. 27

1.27 ULEnableRsaSyncEncryption(SQLCA *) Method. 27

1.28 ULEnableTcpipSynchronization(SQLCA *) Method. 28

1.29 ULEnableZlibSyncCompression(SQLCA *) Method. 29

1.30 ULErrorInfoInitFromSqlca(ul_error_info *, SQLCA const *) Method. 29

1.31 ULErrorInfoParameterAt(ul_error_info const *, ul_u_short, char *, size_t) Method. 30

1.32 ULErrorInfoParameterCount(ul_error_info const *) Method. 30

1.33 ULErrorInfoString(ul_error_info const *, char *, size_t) Method. 31

1.34 ULErrorInfoURL(ul_error_info const *, char *, size_t, char const *) Method. 32

1.35 ULGetDatabaseID(SQLCA *) Method. 32

1.36 ULGetDatabaseProperty(SQLCA *, ul_database_property_id, char *, size_t, ul_bool *) Method . . . 33
1.37 ULGetErrorParameter(SQLCA const *, ul_u_long, char *, size_t) Method. . . . 33
1.38 ULGetErrorParameterCount(SQLCA const *) Method. . . . 34
1.39 ULGetIdentity(SQLCA *) Method. . . . 35
1.40 ULGetLastDownloadTime(SQLCA *, char const *, DECL_DATETIME *) Method. . . . 35
1.41 ULGetNotification(SQLCA *, char const *, char *, ul_length, ul_u_long) Method. . . . 36
1.42 ULGetNotificationParameter(SQLCA *, char const *, char const *, char *, ul_length) Method. . . . 37
1.43 ULGetSyncResult(SQLCA *, ul_sync_result *) Method. . . . 38
1.44 ULGlobalAutoincUsage(SQLCA *) Method. . . . 39
1.45 ULGrantConnectTo(SQLCA *, char const *, char const *) Method. . . . 39
1.46 ULInitSyncInfo(ul_sync_info *) Method. . . . 40
1.47 ULIsSynchronizeMessage(ul_u_long) Method. . . . 41
1.48 ULLibraryVersion(void) Method. . . . 42
1.49 ULRegisterForEvent(SQLCA *, char const *, char const *, char const *, ul_bool) Method. . . . 42
1.50 ULResetLastDownloadTime(SQLCA *, char const *) Method. . . . 43
1.51 ULRevokeConnectFrom(SQLCA *, char const *) Method. . . . 44
1.52 ULRollbackPartialDownload(SQLCA *) Method. . . . 44
1.53 ULRSALibraryVersion(void) Method. . . . 45
1.54 ULSendNotification(SQLCA *, char const *, char const *, char const *) Method. . . . 45
1.55 ULSetDatabaseID(SQLCA *, ul_u_long) Method. . . . 46
1.56 ULSetDatabaseOptionString(SQLCA *, ul_database_option_id, char const *) Method. . . . 47
1.57 ULSetDatabaseOptionULong(SQLCA *, ul_database_option_id, ul_u_long) Method. . . . 47
1.58 ULSetErrorCallback(SQLCA *, ul_error_callback_fn_a, ul_void *, char *, size_t) Method. . . . 48
1.59 ULSetSynchronizationCallback(SQLCA *, ul_sync_observer_fn, ul_void *) Method. . . . 48
1.60 ULSetSyncInfo(SQLCA *, char const *, ul_sync_info *) Method. . . . 49
1.61 ULSignalSyncIsComplete() Method. . . . 50
1.62 ULStartSynchronizationDelete(SQLCA *) Method. . . . 50
1.63 ULStaticFini() Method. . . . 50
1.64 ULStaticInit() Method. . . . 51
1.65 ULStopSynchronizationDelete(SQLCA *) Method. . . . 51
1.66 ULSynchronize(SQLCA *, ul_sync_info *) Method. . . . 52
1.67 ULSynchronizeFromProfile(SQLCA *, char const *, char const *, ul_sync_observer_fn, ul_void *) Method. . . . 52
1.68 ULTriggerEvent(SQLCA *, char const *, char const *) Method. . . . 53
1.69 ULTruncateTable(SQLCA *, ul_table_num) Method. . . . 54
1.70 ULValidateDatabaseLowLevelA(SQLCA *, char const *, ul_u_short, ul_validate_callback_fn, void *) Method. . . . 54
1.71 ULValidateDatabaseConnectedA(SQLCA *, char const *, ul_u_short, ul_validate_callback_fn, void *) Method. . . . 55
1.72 ul_database_option_id Enumeration. . . . 56
1.73 ul_database_property_id Enumeration. . . . 57

1.74 ml_file_transfer_info Structure. 59
1.75 ml_file_transfer_status Structure. 61
1.76 mlft_stream_error Structure. 62
1.77 MLFT_STATUS_FLAG_IS_BLOCKING Variable. 63

1 UltraLite Embedded SQL API Reference

Embedded SQL applications support several UltraLite functions.

Use the EXEC SQL INCLUDE SQLCA command to include prototypes for the functions in this chapter.

Header Files

- `mlfiletransfer.h`
- `ulprotos.h`

The UltraLite Embedded SQL API reference is available in the *UltraLite - Embedded SQL API Reference* at https://help.sap.com/viewer/328faa6a0a4f4931b7e14006b07d7a2b/LATEST/en-US.

In this section:

db_fini Method [page 9]
Frees resources used by the UltraLite runtime library.

db_init Method [page 10]
Initializes the UltraLite runtime library.

MLFileDownload(ml_file_transfer_info *) Method [page 10]
Downloads a file from a MobiLink server with the MobiLink interface.

MLFileUpload(ml_file_transfer_info *) Method [page 12]
Uploads a file from a MobiLink server with the MobiLink interface.

MLFiniFileTransferInfo(ml_file_transfer_info *) Method [page 13]
Finalizes any resources allocated in the ml_file_transfer_info structure when it is initialized.

MLFTEnableRsaE2ee(ml_file_transfer_info *) Method [page 13]
Enables you to specify the RSA end-to-end encryption feature.

MLFTEnableRsaEncryption(ml_file_transfer_info *) Method [page 14]
Enables you to specify the RSA encryption feature.

MLFTEnableRsaFipsE2ee(ml_file_transfer_info *) Method [page 14]
Enables you to specify the RSAFIPS end-to-end encryption feature.

MLFTEnableRsaFipsEncryption(ml_file_transfer_info *) Method [page 15]
Enables you to specify the RSAFIPS encryption feature.

MLFTEnableZlibCompression(ml_file_transfer_info *) Method [page 15]
Enables you to specify the ZLIB compression feature.

MLInitFileTransferInfo(ml_file_transfer_info *) Method [page 16]
Initializes the ml_file_transfer_info structure.

ULCancelGetNotification(SQLCA *, char const *) Method [page 17]

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

ULChangeEncryptionKey(SQLCA *, char const *) Method [page 17]
Changes the encryption key for an UltraLite database.

ULCheckpoint(SQLCA *) Method [page 18]
Performs a checkpoint operation, flushing any pending committed transactions to the database.

ULCountUploadRows(SQLCA *, char const *, ul_u_long) Method [page 18]
Counts the number of rows that need to be uploaded for synchronization.

ULCreateDatabase(SQLCA *, char const *, char const *, void *) Method [page 20]
Creates an UltraLite database.

ULCreateNotificationQueue(SQLCA *, char const *, char const *) Method [page 21]
Creates an event notification queue for this connection.

ULDeclareEvent(SQLCA *, char const *) Method [page 22]
Declares an event which can then be registered for and triggered.

ULDeleteAllRows(SQLCA *, ul_table_num) Method [page 23]
Deletes all rows from a table.

ULDestroyNotificationQueue(SQLCA *, char const *) Method [page 23]
Destroys the given event notification queue.

ULEnableAesDBEncryption(SQLCA *) Method [page 24]
Enables AES database encryption.

ULEnableAesFipsDBEncryption(SQLCA *) Method [page 25]
Enables FIPS 140-2 certified AES database encryption.

ULEnableHttpSynchronization(SQLCA *) Method [page 25]
Enables HTTP synchronization.

ULEnableRsaE2ee(SQLCA *) Method [page 26]
Enables RSA end-to-end encryption.

ULEnableRsaFipsE2ee(SQLCA *) Method [page 26]
Enables FIPS 140-2 certified RSA end-to-end encryption.

ULEnableRsaFipsSyncEncryption(SQLCA *) Method [page 27]
Enables RSA FIPS encryption for SSL or TLS streams.

ULEnableRsaSyncEncryption(SQLCA *) Method [page 27]
Enables RSA encryption for SSL or TLS streams.

ULEnableTcpipSynchronization(SQLCA *) Method [page 28]
Enables TCP/IP synchronization.

ULEnableZlibSyncCompression(SQLCA *) Method [page 29]
Enables ZLIB compression for a synchronization stream.

ULErrorInfoInitFromSqlca(ul_error_info *, SQLCA const *) Method [page 29]
Copies the error information from the SQLCA to the ul_error_info object.

ULErrorInfoParameterAt(ul_error_info const *, ul_u_short, char *, size_t) Method [page 30]
Retrieves an error parameter by ordinal.

ULErrorInfoParameterCount(ul_error_info const *) Method [page 30]
Retrieves the number of error parameters.

ULErrorInfoString(ul_error_info const *, char *, size_t) Method [page 31]
Retrieves a description of the error.

ULErrorInfoURL(ul_error_info const *, char *, size_t, char const *) Method [page 32]
Retrieves a URL to the documentation page for this error.

ULGetDatabaseID(SQLCA *) Method [page 32]
Gets the current database ID used for global autoincrement.

ULGetDatabaseProperty(SQLCA *, ul_database_property_id, char *, size_t, ul_bool *) Method [page 33]
Obtains the value of a database property.

ULGetErrorParameter(SQLCA const *, ul_u_long, char *, size_t) Method [page 33]
Retrieve error parameter via an ordinal parameter number.

ULGetErrorParameterCount(SQLCA const *) Method [page 34]
Obtains a count of the number of error parameters.

ULGetIdentity(SQLCA *) Method [page 35]
Gets the @identity value.

ULGetLastDownloadTime(SQLCA *, char const *, DECL_DATETIME *) Method [page 35]
Obtains the last time a specified publication was downloaded.

ULGetNotification(SQLCA *, char const *, char *, ul_length, ul_u_long) Method [page 36]
Reads an event notification.

ULGetNotificationParameter(SQLCA *, char const *, char const *, char *, ul_length) Method [page 37]
Gets a parameter for the event notification just read by the ULGetNotification method.

ULGetSyncResult(SQLCA *, ul_sync_result *) Method [page 38]
Gets the result of the last synchronization.

ULGlobalAutoincUsage(SQLCA *) Method [page 39]
Obtains the percent of the default values used in all the columns that have global autoincrement defaults.

ULGrantConnectTo(SQLCA *, char const *, char const *) Method [page 39]
Grants access to an UltraLite database for a new or existing user ID with the given password.

ULInitSyncInfo(ul_sync_info *) Method [page 40]
Initializes the synchronization information structure.

ULIsSynchronizeMessage(ul_u_long) Method [page 41]
Checks a message to see if it is a synchronization message from the MobiLink provider for ActiveSync, so that code to handle such a message can be called.

ULLibraryVersion(void) Method [page 42]
Returns the version number of the UltraLite runtime library.

ULRegisterForEvent(SQLCA *, char const *, char const *, char const *, ul_bool) Method [page 42]
Registers or unregisters a queue to receive notifications of an event.

ULResetLastDownloadTime(SQLCA *, char const *) Method [page 43]
Resets the last download time of a publication so that the application resynchronizes previously downloaded data.

ULRevokeConnectFrom(SQLCA *, char const *) Method [page 44]
Revokes access from an UltraLite database for a user ID.

ULRollbackPartialDownload(SQLCA *) Method [page 44]

Rolls back the changes from a failed synchronization.

ULRSALibraryVersion(void) Method [page 45]
Returns the version number of the RSA encryption library.

ULSendNotification(SQLCA *, char const *, char const *, char const *) Method [page 45]
Sends a notification to all queues matching the given name.

ULSetDatabaseID(SQLCA *, ul_u_long) Method [page 46]
Sets the database identification number.

ULSetDatabaseOptionString(SQLCA *, ul_database_option_id, char const *) Method [page 47]
Sets a database option from a string value.

ULSetDatabaseOptionULong(SQLCA *, ul_database_option_id, ul_u_long) Method [page 47]
Sets a numeric database option.

ULSetErrorCallback(SQLCA *, ul_error_callback_fn_a, ul_void *, char *, size_t) Method [page 48]
Sets the callback to be invoked when an error occurs.

ULSetSynchronizationCallback(SQLCA *, ul_sync_observer_fn, ul_void *) Method [page 48]
Sets the callback to be invoked while performing a synchronization.

ULSetSyncInfo(SQLCA *, char const *, ul_sync_info *) Method [page 49]
Creates a synchronization profile using the given name based on the given ul_sync_info structure.

ULSignalSyncIsComplete() Method [page 50]
Indicates that processing a synchronization message is complete.

ULStartSynchronizationDelete(SQLCA *) Method [page 50]
Sets START SYNCHRONIZATION DELETE for this connection.

ULStaticFini() Method [page 50]
Performs finalization of the UltraLite runtime for embedded SQL applications.

ULStaticInit() Method [page 51]
Performs initialization of the UltraLite runtime for embedded SQL applications.

ULStopSynchronizationDelete(SQLCA *) Method [page 51]
Sets STOP SYNCHRONIZATION DELETE for this connection.

ULSynchronize(SQLCA *, ul_sync_info *) Method [page 52]
Initiates synchronization in an UltraLite application.

ULSynchronizeFromProfile(SQLCA *, char const *, char const *, ul_sync_observer_fn, ul_void *) Method [page 52]
Synchronizes the database using the given profile and merge parameters.

ULTriggerEvent(SQLCA *, char const *, char const *) Method [page 53]
Trigger a user-defined event (and send notification to all registered queues).

ULTruncateTable(SQLCA *, ul_table_num) Method [page 54]
Truncates the table and temporarily activates the STOP SYNCHRONIZATION DELETE statement.

ULValidateDatabaseLowLevelA(SQLCA *, char const *, ul_u_short, ul_validate_callback_fn, void *) Method [page 54]
Validates the low-level structure of the specified database.

ULValidateDatabaseConnectedA(SQLCA *, char const *, ul_u_short, ul_validate_callback_fn, void *) Method [page 55]
Validates the database on the current connection.

ul_database_option_id Enumeration [page 56]
    Specifies possible database options that users can set.

ul_database_property_id Enumeration [page 57]
    Specifies possible database properties that users can retrieve.

ml_file_transfer_info Structure [page 59]
    A structure containing the parameters to the file upload/download.

ml_file_transfer_status Structure [page 61]
    A structure containing status/progress information while the file upload/download is in progress.

mlft_stream_error Structure [page 62]
    A structure containing status/progress information while the file upload or download is in progress.

MLFT_STATUS_FLAG_IS_BLOCKING Variable [page 63]
    Defines a bit set in the ml_file_transfer_status.flags field to indicate that the file transfer is blocked awaiting a response from the MobiLink server.

1.1 db_fini Method

Frees resources used by the UltraLite runtime library.

Syntax

```
unsigned short db_fini( SQLCA * sqlca );
```

Returns

- 0 if an error occurs during processing. The error code is set in SQLCA.
- Non-zero if there are no errors.

Remarks

You must not make any other UltraLite library call or execute any Embedded SQL command after db_fini is called.

Call db_fini once for each SQLCA being used.

1.2 db_init Method

Initializes the UltraLite runtime library.

Syntax

```
unsigned short db_init( SQLCA * sqlca );
```

Returns

- 0 if an error occurs during processing (for example, during initialization of the persistent store). The error code is set in SQLCA.
- Non-zero if there are no errors. You can begin using Embedded SQL commands and functions.

Remarks

You must call this function before you make any other UltraLite library call, and before you execute any Embedded SQL command.

Usually you should only call this function once, passing the address of the global sqlca variable (as defined in the `sqlca.h` header file). If you have multiple execution paths in your application, you can use more than one db_init call, as long as each one has a separate sqlca pointer. This separate SQLCA pointer can be a user-defined one, or could be a global SQLCA that has been freed using db_fini.

In multithreaded applications, each thread must call db_init to obtain a separate SQLCA. Carry out subsequent connections and transactions that use this SQLCA on a single thread.

Initializing the SQLCA also resets any settings from previously called ULEnable functions. If you re-initialize a SQLCA, you must issue any ULEnable functions the application requires.

1.3 MLFileDownload(ml_file_transfer_info *) Method

Downloads a file from a MobiLink server with the MobiLink interface.

Syntax

```
public bool MLFileDownload (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Remarks

You must set the source location of the file to be transferred. This location must be specified as a MobiLink user's directory on the MobiLink server (or in the default directory on that server). You can also set the intended target location and file name of the file.

For example, you can program your application to download a new or replacement database from the MobiLink server. You can customize the file for specific users, since the first location that is searched is a specific user's subdirectory. You can also maintain a default file in the root folder on the server, since that location is used if the specified file is not found in the user's folder.

Example

The following example illustrates how to use the MLFileDownload method:

```
ml_file_transfer_info info;
MLInitFileTransferInfo( &info );
MLFTEnableZlibCompression( &info );
info.filename = "myfile";
info.username = "user1";
info.password = "pwd";
info.version = "ver1";
info.stream = "HTTP";
info.stream_parms = "host=myhost.com;compression=zlib";
if( ! MLFileDownload( &info ) ) {
       // file download failed
}
MLFiniFileTransferInfo( &info );
```

Related Information

ml_file_transfer_info Structure [page 59]

1.4 MLFileUpload(ml_file_transfer_info *) Method

Uploads a file from a MobiLink server with the MobiLink interface.

Syntax

```
public bool MLFileUpload (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Remarks

You must set the source location of the file to be transferred. This location must be specified as a MobiLink user's directory on the MobiLink server (or in the default directory on that server). You can also set the intended target location and file name of the file.

For example, you can program your application to upload a new or replacement database from the MobiLink server. You can customize the file for specific users, since the first location that is searched is a specific user's subdirectory. You can also maintain a default file in the root folder on the server, since that location is used if the specified file is not found in the user's folder.

Example

The following example illustrates how to use the MLFileUpload method:

```
ml_file_transfer_info info;
MLInitFileTransferInfo( &info );
MLFTEnableZlibCompression( &info );
info.filename = "myfile";
info.username = "user1";
info.password = "pwd";
info.version = "ver1";
info.stream = "HTTP";
info.stream_parms = "host=myhost.com;compression=zlib";
if( ! MLFileUpload( &info ) ) {
          // file upload failed
}
MLFiniFileTransferInfo( &info );
```

Related Information

ml_file_transfer_info Structure [page 59]

1.5 MLFiniFileTransferInfo(ml_file_transfer_info *) Method

Finalizes any resources allocated in the ml_file_transfer_info structure when it is initialized.

Syntax

```
public void MLFiniFileTransferInfo (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Remarks

This method should be called after the file upload/download has completed.

Related Information

ml_file_transfer_info Structure [page 59]

1.6 MLFTEnableRsaE2ee(ml_file_transfer_info *) Method

Enables you to specify the RSA end-to-end encryption feature.

Syntax

```
public void MLFTEnableRsaE2ee (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Related Information

ml_file_transfer_info Structure [page 59]

1.7 MLFTEnableRsaEncryption(ml_file_transfer_info *) Method

Enables you to specify the RSA encryption feature.

Syntax

```
public void MLFTEnableRsaEncryption (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Related Information

ml_file_transfer_info Structure [page 59]

1.8 MLFTEnableRsaFipsE2ee(ml_file_transfer_info *) Method

Enables you to specify the RSAFIPS end-to-end encryption feature.

Syntax

```
public void MLFTEnableRsaFipsE2ee (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Related Information

ml_file_transfer_info Structure [page 59]

1.9 MLFTEnableRsaFipsEncryption(ml_file_transfer_info *) Method

Enables you to specify the RSAFIPS encryption feature.

Syntax

```
public void MLFTEnableRsaFipsEncryption (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Related Information

ml_file_transfer_info Structure [page 59]

1.10 MLFTEnableZlibCompression(ml_file_transfer_info *) Method

Enables you to specify the ZLIB compression feature.

Syntax

```
public void MLFTEnableZlibCompression (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Related Information

ml_file_transfer_info Structure [page 59]

1.11 MLInitFileTransferInfo(ml_file_transfer_info *) Method

Initializes the ml_file_transfer_info structure.

Syntax

```
public bool MLInitFileTransferInfo (ml_file_transfer_info * info)
```

Parameters

info A structure containing the file transfer information.

Remarks

This method should be called before starting the file upload/download.

Related Information

ml_file_transfer_info Structure [page 59]

1.12 ULCancelGetNotification(SQLCA *, char const *) Method

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

Syntax

```
public ul_u_long ULCancelGetNotification (
    SQLCA * sqlca,
    char const * queue_name
)
```

Parameters

sqlca A pointer to the SQLCA.

queue_name The name of the queue.

Returns

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

1.13 ULChangeEncryptionKey(SQLCA *, char const *) Method

Changes the encryption key for an UltraLite database.

Syntax

```
public ul_bool ULChangeEncryptionKey (
    SQLCA * sqlca,
    char const * new_key
)
```

Parameters

sqlca A pointer to the SQLCA.

new_key The new encryption key.

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 this 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.

1.14 ULCheckpoint(SQLCA *) Method

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

Syntax

```
public ul_ret_void ULCheckpoint (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Remarks

Any current transaction is not committed by calling this method. This method is used in conjunction with deferring automatic transaction checkpoints as a performance enhancement.

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

1.15 ULCountUploadRows(SQLCA *, char const *, ul_u_long) Method

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

Syntax

```
public ul_u_long ULCountUploadRows (
    SQLCA * sqlca,
    char const * pub_list,
    ul_u_long threshold
)
```

Parameters

sqlca A pointer to the SQL.

pub_list 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 the pub_list string 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, the method counts all the 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 = ULCountUploadRows( sqlca, UL_SYNC_ALL, 0 );
```

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

```
count = ULCountUploadRows( sqlca, UL_TEXT("PUB1,PUB2"), 1000 );
```

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

```
count = ULCountUploadRows( sqlca, UL_TEXT("PUB1,PUB2"), 1 );
```

1.16 ULCreateDatabase(SQLCA *, char const *, char const *, void *) Method

Creates an UltraLite database.

Syntax

```
public ul_bool ULCreateDatabase (
    SQLCA * sqlca,
    char const * connect_parms,
    char const * create_parms,
    void * reserved
)
```

Parameters

sqlca A pointer to the initialized SQLCA.

connect_parms 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.

create_parms A semicolon-separated string of creation parameters, a set of keyword=value pairs, such as page_size=2048;kdf_iterations=100.

reserved This parameter is reserved for future use.

Returns

ul_true if database was successfully created; otherwise, returns ul_false. Typically ul_false is caused by an invalid file name or denied access.

Remarks

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

The connect_parms parameter is a list of connection parameters that are applicable whenever the database is accessed. Some examples include file name, user ID, password, or optional encryption key.

The create_parms parameter is a list of parameters that are only relevant when creating a database. Some examples include obfuscation, page-size, and time and date format).

Applications can call this method after initializing the SQLCA.

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

```
if( ULCreateDatabase( &sqlca,
   "DBF=path/myfile.udb;key=anicelongpassphrase",
   ULGetCollation_1250LATIN2(),
   "checksum_level=2",
   NULL )
{
   // success
};
```

1.17 ULCreateNotificationQueue(SQLCA *, char const *, char const *) Method

Creates an event notification queue for this connection.

Syntax

```
public ul_bool ULCreateNotificationQueue (
    SQLCA * sqlca,
    char const * name,
    char const * parameters
)
```

Parameters

sqlca A pointer to the SQLCA.
name The name for the new queue.
parameters Currently unused. 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 ULRegisterForEvent method if no queue is specified. This call fails with an error if the name already exists or isn't valid.

1.18 ULDeclareEvent(SQLCA *, char const *) Method

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

Syntax

```
public ul_bool ULDeclareEvent (
    SQLCA * sqlca,
    char const * event_name
)
```

Parameters

sqlca A pointer to the SQLCA.
event_name 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 function declares user-defined events. User-defined events are triggered with ULTriggerEvent method. The event name must be unique. Names are case insensitive.

Related Information

ULTriggerEvent(SQLCA *, char const *, char const *) Method [page 53]

1.19 ULDeleteAllRows(SQLCA *, ul_table_num) Method

Deletes all rows from a table.

Syntax

```
public ul_ret_void ULDeleteAllRows (
    SQLCA * sqlca,
    ul_table_num number
)
```

Parameters

sqlca A pointer to the SQLCA.

number The ID of the table to truncate.

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.

Note

Any uncommitted inserts from other connections are not deleted. Also, any uncommitted deletes from other connections are not deleted, if the other connection does 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.

1.20 ULDestroyNotificationQueue(SQLCA *, char const *) Method

Destroys the given event notification queue.

Syntax

```
public ul_bool ULDestroyNotificationQueue (
    SQLCA * sqlca,
    char const * name
)
```

Parameters

sqlca A pointer to the SQLCA.
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.

1.21 ULEnableAesDBEncryption(SQLCA *) Method

Enables AES database encryption.

Syntax

```
public ul_ret_void ULEnableAesDBEncryption (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the initialized SQLCA.

Remarks

You can use this method in C++ API applications and embedded SQL applications. You must call this method before calling the ULInitDatabaseManager method.

Note

Calling this method causes the encryption routines to be included in the application and increases the size of the application code.

1.22 ULEnableAesFipsDBEncryption(SQLCA *) Method

Enables FIPS 140-2 certified AES database encryption.

Syntax

```
public ul_ret_void ULEnableAesFipsDBEncryption (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the initialized SQLCA.

Remarks

i Note

Calling this method causes the appropriate routines to be included in the application and increases the size of the application code.

You can use this method in C++ API applications and embedded SQL applications. You must call this method before the Synchronize method. If you attempt to synchronize without a preceding call to enable the synchronization type, the SQLE_METHOD_CANNOT_BE_CALLED error occurs.

Related Information

ULEnableAesDBEncryption(SQLCA *) Method [page 24]

1.23 ULEnableHttpSynchronization(SQLCA *) Method

Enables HTTP synchronization.

Syntax

```
public ul_ret_void ULEnableHttpSynchronization (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Remarks

You can use this method in C++ API applications and embedded SQL applications. You must call this method before the Synchronize method. If you attempt to synchronize without a preceding call to enable the synchronization type, the SQLE_METHOD_CANNOT_BE_CALLED error occurs.

1.24 ULEnableRsaE2ee(SQLCA *) Method

Enables RSA end-to-end encryption.

Syntax

```
public ul_ret_void ULEnableRsaE2ee (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

1.25 ULEnableRsaFipsE2ee(SQLCA *) Method

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

Syntax

```
public ul_ret_void ULEnableRsaFipsE2ee (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

1.26 ULEnableRsaFipsSyncEncryption(SQLCA *) Method

Enables RSA FIPS encryption for SSL or TLS streams.

Syntax

```
public ul_ret_void ULEnableRsaFipsSyncEncryption (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Remarks

This is required when setting a stream parameter to TLS or HTTPS.

You can use this method in C++ API applications and embedded SQL applications. You must call this method before the Synchronize method. If you attempt to synchronize without a preceding call to enable the synchronization type, the SQLE_METHOD_CANNOT_BE_CALLED error occurs.

Related Information

ULEnableRsaSyncEncryption(SQLCA *) Method [page 27]

1.27 ULEnableRsaSyncEncryption(SQLCA *) Method

Enables RSA encryption for SSL or TLS streams.

Syntax

```
public ul_ret_void ULEnableRsaSyncEncryption (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Remarks

This is required when setting a stream parameter to TLS or HTTPS.

You can use this method in C++ API applications and embedded SQL applications. You must call this method before the Synchronize method. If you attempt to synchronize without a preceding call to enable the synchronization type, the SQLE_METHOD_CANNOT_BE_CALLED error occurs.

Related Information

ULEnableRsaFipsSyncEncryption(SQLCA *) Method [page 27]

1.28 ULEnableTcpipSynchronization(SQLCA *) Method

Enables TCP/IP synchronization.

Syntax

```
public ul_ret_void ULEnableTcpipSynchronization (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Remarks

You can use this method in C++ API applications and embedded SQL applications. You must call this method before the Synchronize method. If you attempt to synchronize without a preceding call to enable the synchronization type, the SQLE_METHOD_CANNOT_BE_CALLED error occurs.

1.29 ULEnableZlibSyncCompression(SQLCA *) Method

Enables ZLIB compression for a synchronization stream.

Syntax

```
public ul_ret_void ULEnableZlibSyncCompression (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the initialized SQLCA.

Remarks

You can use this method in C++ API applications and embedded SQL applications. You must call this method before calling the Synchronize method. If you attempt to synchronize without a preceding call to enable the synchronization type, the SQLE_METHOD_CANNOT_BE_CALLED error occurs.

1.30 ULErrorInfoInitFromSqlca(ul_error_info *, SQLCA const *) Method

Copies the error information from the SQLCA to the ul_error_info object.

Syntax

```
public void ULErrorInfoInitFromSqlca (
    ul_error_info * errinf,
    SQLCA const * sqlca
)
```

Parameters

sqlca A pointer to the SQLCA.
errinf The ul_error_info object.

1.31 ULErrorInfoParameterAt(ul_error_info const *, ul_u_short, char *, size_t) Method

Retrieves an error parameter by ordinal.

Syntax

```
public size_t ULErrorInfoParameterAt (
    ul_error_info const * errinf,
    ul_u_short parmNo,
    char * buffer,
    size_t bufferSize
)
```

Parameters

errinf The ul_error_info object.
parmNo The 1-based parameter ordinal.
buffer The buffer to receive parameter string.
bufferSize The size of the buffer.

Returns

The size, in bytes, required to store the parameter, or zero if the ordinal isn't valid. If the return value is larger than the bufferSize value, the parameter was truncated.

1.32 ULErrorInfoParameterCount(ul_error_info const *) Method

Retrieves the number of error parameters.

Syntax

```
public ul_u_short ULErrorInfoParameterCount (ul_error_info const * errinf)
```

Parameters

errinf The ul_error_info object.

Returns

The number of error parameters.

1.33 ULErrorInfoString(ul_error_info const *, char *, size_t) Method

Retrieves a description of the error.

Syntax

```
public size_t ULErrorInfoString (
    ul_error_info const * errinf,
    char * buffer,
    size_t bufferSize
)
```

Parameters

errinf The ul_error_info object.
buffer The buffer to receive the error description.
bufferSize The size, in bytes, of the buffer.

Returns

The size, in bytes, required to store the string. If the return value is larger than the len value, the string was truncated.

1.34 ULErrorInfoURL(ul_error_info const *, char *, size_t, char const *) Method

Retrieves a URL to the documentation page for this error.

Syntax

```
public size_t ULErrorInfoURL (
    ul_error_info const * errinf,
    char * buffer,
    size_t bufferSize,
    char const * reserved
)
```

Parameters

errinf The ul_error_info object.

buffer The buffer to receive the URL.

bufferSize The size, in bytes, of the buffer.

reserved Reserved for future use.

Returns

The size, in bytes, required to store the URL. If the return value is larger than the len value, the URL was truncated.

1.35 ULGetDatabaseID(SQLCA *) Method

Gets the current database ID used for global autoincrement.

Syntax

```
public ul_u_long ULGetDatabaseID (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Returns

The value set by the last call to the SetDatabaseID method, or UL_INVALID_DATABASE_ID if the ID was never set.

1.36 ULGetDatabaseProperty(SQLCA *, ul_database_property_id, char *, size_t, ul_bool *) Method

Obtains the value of a database property.

Syntax

```
public void ULGetDatabaseProperty (
    SQLCA * sqlca,
    ul_database_property_id id,
    char * dst,
    size_t buffer_size,
    ul_bool * null_indicator
)
```

Parameters

sqlca A pointer to the SQLCA.
id The identifier for the database property.
dst A character array to store the value of the property.
buffer_size The size of the character array dst.
null_indicator An indicator that the database parameter is null.

1.37 ULGetErrorParameter(SQLCA const *, ul_u_long, char *, size_t) Method

Retrieve error parameter via an ordinal parameter number.

Syntax

```
public size_t ULGetErrorParameter (
    SQLCA const * sqlca,
    ul_u_long parm_num,
    char * buffer,
    size_t size
)
```

Parameters

sqlca A pointer to the SQLCA.
parm_num The ordinal parameter number.
buffer A pointer to a buffer that contains the error parameter.
size The size, in bytes, of the buffer.

Returns

This method returns the number of characters copied to the supplied buffer.

Related Information

ULGetErrorParameterCount(SQLCA const *) Method [page 34]

1.38 ULGetErrorParameterCount(SQLCA const *) Method

Obtains a count of the number of error parameters.

Syntax

```
public ul_u_long ULGetErrorParameterCount (SQLCA const * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Returns

The number of error parameters. Unless the result is zero, values from 1 through this result can be used to call the ULGetErrorParameter method to retrieve the corresponding error parameter value.

Related Information

ULGetErrorParameter(SQLCA const *, ul_u_long, char *, size_t) Method [page 33]

1.39 ULGetIdentity(SQLCA *) Method

Gets the @identity value.

Syntax

```
public ul_u_big ULGetIdentity (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Returns

The last value inserted into an autoincrement or global autoincrement column.

1.40 ULGetLastDownloadTime(SQLCA *, char const *, DECL_DATETIME *) Method

Obtains the last time a specified publication was downloaded.

Syntax

```
public ul_bool ULGetLastDownloadTime (
    SQLCA * sqlca,
    char const * pub_name,
    DECL_DATETIME * value
)
```

Parameters

sqlca A pointer to the SQLCA.

pub_name A string containing a publication name for which the last download time is retrieved.

value A pointer to the DECL_DATETIME structure to be populated. For example, the value of January 1, 1990 indicates that the publication has yet to be synchronized.

Returns

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

Remarks

The following call populates the dt structure with the date and time that the UL_PUB_PUB1 publication was downloaded:

```
DECL_DATETIME dt;
ret = ULGetLastDownloadTime( &sqlca, UL_TEXT("UL_PUB_PUB1"), &dt );
```

1.41 ULGetNotification(SQLCA *, char const *, char *, ul_length, ul_u_long) Method

Reads an event notification.

Syntax

```
public ul_bool ULGetNotification (
    SQLCA * sqlca,
    char const * queue_name,
    char * event_name_buf,
    ul_length event_name_buf_len,
    ul_u_long wait_ms
)
```

Parameters

sqlca A pointer to the SQLCA.

queue_name The queue to read or NULL for the default connection queue.

event_name_buf A buffer to hold the name of the event.

event_name_buf_len The size of the buffer in bytes.

wait_ms The time, in milliseconds, to wait (block) before returning.

Returns

True on success; otherwise, returns false.

Remarks

This call blocks until a notification is received or until the given wait period expires. Pass UL_READ_WAIT_INFINITE to the wait_ms parameter to wait indefinitely. To cancel a wait, send another notification to the given queue or use the ULCancelGetNotification method. After reading a notification, use the ULGetNotificationParameter method to retrieve additional parameters by name.

Related Information

ULCancelGetNotification(SQLCA *, char const *) Method [page 17]

ULGetNotificationParameter(SQLCA *, char const *, char const *, char *, ul_length) Method [page 37]

1.42 ULGetNotificationParameter(SQLCA *, char const *, char const *, char *, ul_length) Method

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

Syntax

```
public ul_bool ULGetNotificationParameter (
    SQLCA * sqlca,
    char const * queue_name,
    char const * parameter_name,
    char * value_buf,
    ul_length value_buf_len
)
```

Parameters

sqlca A pointer to the SQLCA.

queue_name The queue to read or NULL for default connection queue.
parameter_name The name of the parameter to read (or "*").
value_buf A buffer to hold the parameter value.
value_buf_len The size of the buffer in bytes.

Returns

True on success; otherwise, returns false.

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.

1.43 ULGetSyncResult(SQLCA *, ul_sync_result *) Method

Gets the result of the last synchronization.

Syntax

```
public ul_bool ULGetSyncResult (
    SQLCA * sqlca,
    ul_sync_result * sync_result
)
```

Parameters

sqlca A pointer to the SQLCA.
sync_result A pointer to the ul_sync_result structure that holds the synchronization results.

Returns

True on success; otherwise, returns false.

1.44 ULGlobalAutoincUsage(SQLCA *) Method

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

Syntax

```
public ul_u_short ULGlobalAutoincUsage (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

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.

Related Information

ULSetDatabaseID(SQLCA *, ul_u_long) Method [page 46]

1.45 ULGrantConnectTo(SQLCA *, char const *, char const *) Method

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

Syntax

```
public ul_ret_void ULGrantConnectTo (
    SQLCA * sqlca,
    char const * uid,
    char const * pwd
)
```

Parameters

sqlca A pointer to the SQLCA.

uid A character array that holds the user ID.

pwd A character array that holds the password for the user ID.

Remarks

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

Related Information

ULRevokeConnectFrom(SQLCA *, char const *) Method [page 44]

1.46 ULInitSyncInfo(ul_sync_info *) Method

Initializes the synchronization information structure.

Syntax

```
public ul_ret_void ULInitSyncInfo (ul_sync_info * info)
```

Parameters

info A synchronization structure.

1.47 ULIsSynchronizeMessage(ul_u_long) Method

Checks a message to see if it is a synchronization message from the MobiLink provider for ActiveSync, so that code to handle such a message can be called.

Syntax

```
public ul_bool ULIsSynchronizeMessage (ul_u_long number)
```

Remarks

When the processing of a synchronization message is complete, the ULSignalSyncIsComplete method should be called.

You should include a call to this method in the WindowProc function of your application. This applies to Windows Mobile for ActiveSync.

The following code snippet illustrates how to use the ULIsSynchronizeMessage method to handle a synchronization message:

```
LRESULT CALLBACK WindowProc( HWND hwnd,
        UINT uMsg,
        WPARAM wParam,
        LPARAM lParam )
{
 if( ULIsSynchronizeMessage( uMsg ) ) {
   // execute synchronization code
   if( wParam == 1 ) DestroyWindow( hWnd );
   return 0;
 }
 switch( uMsg ) {
 // code to handle other windows messages
 default:
   return DefWindowProc( hwnd, uMsg, wParam, lParam );
 }
 return 0;
}
```

Related Information

ULSignalSyncIsComplete() Method [page 50]

1.48 ULLibraryVersion(void) Method

Returns the version number of the UltraLite runtime library.

Syntax

```
public char const * ULLibraryVersion (void)
```

Returns

The version number of the UltraLite runtime library.

1.49 ULRegisterForEvent(SQLCA *, char const *, char const *, char const *, ul_bool) Method

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

Syntax

```
public ul_bool ULRegisterForEvent (
    SQLCA * sqlca,
    char const * event_name,
    char const * object_name,
    char const * queue_name,
    ul_bool register_not_unreg
)
```

Parameters

sqlca A pointer to the SQLCA.
event_name The system- or user-defined event to register for.
object_name The object to which the event applies, such as a table name.
queue_name The connection queue name. NULL denotes the default connection queue.
register_not_unreg True to register; false to unregister.

Returns

True if the registration succeeded; 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 ULSendNotification 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.

1.50 ULResetLastDownloadTime(SQLCA *, char const *) Method

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

Syntax

```
public ul_ret_void ULResetLastDownloadTime (
    SQLCA * sqlca,
    char const * pub_list
)
```

Parameters

sqlca A pointer to the SQLCA

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

Remarks

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

```
ULResetLastDownloadTime( &sqlca, UL_TEXT("*") );
```

1.51 ULRevokeConnectFrom(SQLCA *, char const *) Method

Revokes access from an UltraLite database for a user ID.

Syntax

```
public ul_ret_void ULRevokeConnectFrom (
    SQLCA * sqlca,
    char const * uid
)
```

Parameters

sqlca A pointer to the SQLCA.

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

1.52 ULRollbackPartialDownload(SQLCA *) Method

Rolls back the changes from a failed synchronization.

Syntax

```
public ul_ret_void ULRollbackPartialDownload (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

Remarks

When a communication error occurs during the download phase of synchronization, UltraLite can apply the downloaded changes, so that the application can resume the synchronization from the place it was interrupted. If the download changes are not needed (the user or application does not want to resume the download at this point), the ULRollbackPartialDownload method rolls back the failed download transaction.

1.53 ULRSALibraryVersion(void) Method

Returns the version number of the RSA encryption library.

Syntax

```
public char const * ULRSALibraryVersion (void)
```

Returns

The version number of the RSA encryption library.

1.54 ULSendNotification(SQLCA *, char const *, char const *, char const *) Method

Sends a notification to all queues matching the given name.

Syntax

```
public ul_u_long ULSendNotification (
    SQLCA * sqlca,
    char const * queue_name,
    char const * event_name,
    char const * parameters
)
```

Parameters

sqlca A pointer to the SQLCA.
queue_name The connection queue name. NULL indicates the default connection queue.

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

parameters Currently unused. Set to NULL.

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 ULGetNotificationParameter method.

Related Information

ULGetNotificationParameter(SQLCA *, char const *, char const *, char *, ul_length) Method [page 37]

1.55 ULSetDatabaseID(SQLCA *, ul_u_long) Method

Sets the database identification number.

Syntax

```
public ul_ret_void ULSetDatabaseID (
    SQLCA * sqlca,
    ul_u_long value
)
```

Parameters

sqlca A pointer to the SQLCA.

value A positive integer that uniquely identifies a particular database in a replication or synchronization setup.

Related Information

ULGlobalAutoincUsage(SQLCA *) Method [page 39]

1.56 ULSetDatabaseOptionString(SQLCA *, ul_database_option_id, char const *) Method

Sets a database option from a string value.

Syntax

```
public void ULSetDatabaseOptionString (
    SQLCA * sqlca,
    ul_database_option_id id,
    char const * value
)
```

Parameters

sqlca A pointer to the SQLCA.
id The identifier for the database option to be set.
value The value of the database option.

1.57 ULSetDatabaseOptionULong(SQLCA *, ul_database_option_id, ul_u_long) Method

Sets a numeric database option.

Syntax

```
public void ULSetDatabaseOptionULong (
    SQLCA * sqlca,
    ul_database_option_id id,
    ul_u_long value
)
```

Parameters

sqlca A pointer to the SQLCA.
id The identifier for the database option to be set.
value The value of the database option.

1.58 ULSetErrorCallback(SQLCA *, ul_error_callback_fn_a, ul_void *, char *, size_t) Method

Sets the callback to be invoked when an error occurs.

Syntax

```
public ul_ret_void ULSetErrorCallback (
    SQLCA * sqlca,
    ul_error_callback_fn_a callback,
    ul_void * user_data,
    char * buffer,
    size_t len
)
```

Parameters

sqlca A pointer to the SQLCA.
callback The callback function.
user_data User context information passed to the callback.
buffer A user-supplied buffer that contains the error parameters when the callback is invoked.
len The size, in bytes, of the buffer.

1.59 ULSetSynchronizationCallback(SQLCA *, ul_sync_observer_fn, ul_void *) Method

Sets the callback to be invoked while performing a synchronization.

Syntax

```
public ul_ret_void ULSetSynchronizationCallback (
    SQLCA * sqlca,
    ul_sync_observer_fn callback,
    ul_void * user_data
)
```

Parameters

sqlca A pointer to the SQLCA.
callback The callback.
user_data User context information that is passed to the callback.

1.60 ULSetSyncInfo(SQLCA *, char const *, ul_sync_info *) Method

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

Syntax

```
public ul_bool ULSetSyncInfo (
    SQLCA * sqlca,
    char const * profile_name,
    ul_sync_info * sync_info
)
```

Parameters

sqlca A pointer to the SQLCA.
profile_name The name of the synchronization profile.
sync_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.

1.61 ULSignalSyncIsComplete() Method

Indicates that processing a synchronization message is complete.

Syntax

```
public ul_ret_void ULSignalSyncIsComplete ()
```

Remarks

Applications that are registered with the Microsoft ActiveSync provider need to call this method in their WNDPROC when processing a synchronization message is complete.

1.62 ULStartSynchronizationDelete(SQLCA *) Method

Sets START SYNCHRONIZATION DELETE for this connection.

Syntax

```
public ul_ret_void ULStartSynchronizationDelete (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA.

1.63 ULStaticFini() Method

Performs finalization of the UltraLite runtime for embedded SQL applications.

Syntax

```
public void ULStaticFini ()
```

Remarks

This method should be called once and only once per application, after which no other UltraLite method should be called.

1.64 ULStaticInit() Method

Performs initialization of the UltraLite runtime for embedded SQL applications.

Syntax

```
public void ULStaticInit ()
```

Remarks

This method should be called once and only once per application, before any other UltraLite methods have been called.

1.65 ULStopSynchronizationDelete(SQLCA *) Method

Sets STOP SYNCHRONIZATION DELETE for this connection.

Syntax

```
public ul_bool ULStopSynchronizationDelete (SQLCA * sqlca)
```

Parameters

sqlca A pointer to the SQLCA

Returns

True on success; otherwise, returns false.

1.66 ULSynchronize(SQLCA *, ul_sync_info *) Method

Initiates synchronization in an UltraLite application.

Syntax

```
public ul_ret_void ULSynchronize (
    SQLCA * sqlca,
    ul_sync_info * info
)
```

Parameters

sqlca A pointer to the SQLCA.

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

Remarks

For TCP/IP or HTTP synchronization, the ULSynchronize method initiates synchronization. Errors during synchronization that are not handled by the handle_error script are reported as SQL errors. Application programs should test the SQLCODE return value of this method.

The following example demonstrates database synchronization:

```
ul_sync_info info;
ULInitSyncInfo( &info );
info.user_name = UL_TEXT( "user_name" );
info.version = UL_TEXT( "test" );
ULSynchronize( &sqlca, &info );
```

1.67 ULSynchronizeFromProfile(SQLCA *, char const *, char const *, ul_sync_observer_fn, ul_void *) Method

Synchronizes the database using the given profile and merge parameters.

Syntax

```
public ul_ret_void ULSynchronizeFromProfile (
    SQLCA * sqlca,
    char const * profile_name,
    char const * merge_parms,
    ul_sync_observer_fn observer,
    ul_void * user_data
)
```

Parameters

sqlca A pointer to the SQLCA.
profile_name The name of the profile to synchronize.
merge_parms Merge parameters for the synchronization.
observer Observer callback to send status updates to.
user_data User context data passed to callback.

Remarks

This method is identical to executing the SYNCHRONIZE statement.

1.68 ULTriggerEvent(SQLCA *, char const *, char const *) Method

Trigger a user-defined event (and send notification to all registered queues).

Syntax

```
public ul_u_long ULTriggerEvent (
    SQLCA * sqlca,
    char const * event_name,
    char const * parameters
)
```

Parameters

sqlca A pointer for the SQLCA.
event_name The system or user-defined event to register for.
parameters Currently unused. Set to NULL.

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 the ULGetNotificationParameter method.

Related Information

ULGetNotificationParameter(SQLCA *, char const *, char const *, char *, ul_length) Method [page 37]

1.69 ULTruncateTable(SQLCA *, ul_table_num) Method

Truncates the table and temporarily activates the STOP SYNCHRONIZATION DELETE statement.

Syntax

```
public ul_ret_void ULTruncateTable (
    SQLCA * sqlca,
    ul_table_num number
)
```

Parameters

sqlca A pointer to the SQLCA.
number The ID of the table to truncate.

1.70 ULValidateDatabaseLowLevelA(SQLCA *, char const *, ul_u_short, ul_validate_callback_fn, void *) Method

Validates the low-level structure of the specified database.

Syntax

```
public ul_bool ULValidateDatabase (
    SQLCA * sqlca,
    char const * start_parms,
    ul_u_short flags,
    ul_validate_callback_fn callback_fn,
    void * user_data
)
```

Parameters

sqlca A pointer to the SQLCA.
connect_parms Connection parameters identifying database to check.
flags `ULVF_` validation flags: specify **`ULVF_DATABASE`**.
callback_fn Function to receive validation information.
user_data User data passed to `callback_fn`.

Returns

True if validate found no problems; false if problems were detected.

Remarks

Use this function to detect low-level corruption in a database which will not start properly for use with the normal ULValidateDatabaseConnectedA().

1.71 ULValidateDatabaseConnectedA(SQLCA *, char const *, ul_u_short, ul_validate_callback_fn, void *) Method

Validates the database on the current connection.

Syntax

```
public ul_bool ULValidateDatabaseTableName (
    SQLCA * sqlca,
    char const * table_name,
    ul_u_short flags,
    ul_validate_callback_fn callback_fn,
    void * user_data
)
```

Parameters

sqlca Active SQLCA pointer.
table_name Name of specific table to validate, or NULL for all tables.
flags `ULVF_` validation flags.
callback_fn Function to receive validation information.

user_data User data passed to `callback_fn`.

Returns

True if validate found no problems; false if problems were detected.

Remarks

To receive information about the validation, implement a callback function and pass the address to this routine.

The flags parameter determines what is validated, and is a combination of the following bit values:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE

And the modifier:

- ULVF_EXPRESS

ULVF_FULL_VALIDATE is shorthand for all validation (but not express).

To limit the validation to a specific table, specify the name with the `table_name` parameter.

1.72 ul_database_option_id Enumeration

Specifies possible database options that users can set.

Syntax

```
enum ul_database_option_id
```

Members

Member name	Description
ul_option_global_database_id	The global database ID is set using an unsigned long integer.
ul_option_ml_remote_id	The remote ID is set using a string.
ul_option_commit_flush_timeout	The database commit flush timeout is set as an integer, representing a time threshold measured in milliseconds.

Member name	Description
ul_option_commit_flush_count	The database commit flush count is set as integer, representing a commit count threshold.
ul_option_isolation_level	The connection isolation level is set as string. (read_committed/read_uncommitted)
ul_option_cache_allocation	Set to resize the database file cache. The value is an integer in the range 0 to 100, representing the amount of cache allocated of the minimum to maximum size range.

Remarks

These database options are used with the ULConnection.SetDatabaseOption method.

1.73 ul_database_property_id Enumeration

Specifies possible database properties that users can retrieve.

Syntax

```
enum ul_database_property_id
```

Members

Member name	Description
ul_property_date_format	Date format. (date_format)
ul_property_date_order	Date order. (date_order)
ul_property_nearest_century	Nearest century. (nearest_century)
ul_property_precision	Precision. (precision)

Member name	Description
ul_property_scale	Scale. (scale)
ul_property_time_format	Time format. (time_format)
ul_property_timestamp_format	Timestamp format. (timestamp_format)
ul_property_timestamp_increment	Timestamp increment. (timestamp_increment)
ul_property_name	Name. (Name)
ul_property_file	File. (File)
ul_property_encryption	Encryption. (Encryption)
ul_property_global_database_id	Global database ID. (global_database_id)
ul_property_ml_remote_id	Remote ID. (ml_remote_id)
ul_property_char_set	Character set. (CharSet)
ul_property_collation	collation sequence. (Collation)
ul_property_page_size	Page size. (PageSize)
ul_property_case_sensitive	CaseSensitive. (CaseSensitive)
ul_property_conn_count	Connection count. (ConnCount)
ul_property_max_hash_size	Default maximum index hash. (MaxHashSize)
ul_property_checksum_level	Database checksum level. (ChecksumLevel)

Member name	Description
ul_property_checkpoint_count	Database checkpoint count. (CheckpointCount)
ul_property_commit_flush_timeout	Database commit flush timeout. (commit_flush_timeout)
ul_property_commit_flush_count	Database commit flush count. (commit_flush_count)
ul_property_isolation_level	Connection isolation level. (isolation_level)
ul_property_timestamp_with_time_zone_format	Timestamp with time zone format. (timestamp_with_time_zone_format)
ul_property_cache_allocation	The current database file cache size, as a percentage value of the minimum to maximum range.
ul_property_partial_download	Indicate presence of partial download (which could be re-sumed). (PartialDownload)
ul_property_upload_unknown	Indicate that the status of our last upload is unknown. (UploadUnknown)

Remarks

These properties are used with the ULConnection.GetDatabaseProperty method.

1.74 ml_file_transfer_info Structure

A structure containing the parameters to the file upload/download.

Syntax

```
typedef struct ml_file_transfer_info
```

Members

All members of ml_file_transfer_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public const char *	filename	The file name to be transferred from the server running MobiLink. MobiLink searches the username sub-directory first, before defaulting to the root directory.
public const char *	local_path	The local path to store the downloaded file. If this parameter is empty (the default), the downloaded file is stored in the current directory. On Windows Mobile, if dest_path is empty, the file is stored in the root () directory of the device. On the desktop, if the dest_path value is empty, the file is stored in the user's current directory.
public const char *	local_filename	The local name for the downloaded file. If this parameter is empty, the value in file name is used.
public const char *	stream	The protocol can be one of: TCPIP, TLS, HTTP, or HTTPS. This field is required.
public const char *	stream_parms	The protocol options for a given stream.
public const char *	username	The MobiLink user name. This field is required.
public const char *	remote_key	The MobiLink remote key.
public const char *	password	The password for the MobiLink user name.
public const char *	version	The MobiLink script version. This field is required.
public ml_file_transfer_observer_fn	observer	A callback can be provided to observe file download progress through the 'observer' field. For more details, see the description of the callback function that follows.
public void *	user_data	The application-specific information made available to the synchronization observer.

Modifier and Type	Variable	Description
public bool	enable_resume	If set to true, the MLFileDownload method resumes a previous download that was interrupted due to a communications error or if it was canceled by the user. If the file on the server is newer than the partial local file, the partial file is discarded and the new version is downloaded from the beginning. The default is true.
public asa_uint8	num_auth_parms	The number of authentication parameters being passed to authentication parameters in MobiLink events.
public const char **	auth_parms	Supplies parameters to authentication parameters in MobiLink events.
public asa_uint16	transferred_file	1 if the file was successfully transferred, and 0 if an error occurs. An error occurs if the file is already up-to-date when MLFileUpload is invoked. In this case, the function returns true rather than false.
public asa_uint16	auth_status	Supplies parameters to authentication parameters in MobiLink events.
public asa_uint32	auth_value	Reports results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client.
public asa_uint16	file_auth_code	Contains the return code of the optional authenticate_file_transfer script on the server.
public mlft_stream_error	error	Contains information about any error that occurs.

1.75 ml_file_transfer_status Structure

A structure containing status/progress information while the file upload/download is in progress.

Syntax

```
typedef struct ml_file_transfer_status
```

Members

All members of ml_file_transfer_status, including inherited members.

Variables

Modifier and Type	Variable	Description
public asa_uint64	file_size	Indicates the total size, in bytes, of the file being downloaded.
public asa_uint64	bytes_transferred	Indicates how much of the file has been downloaded so far, including previous synchronizations, if the download is resumed.
public asa_uint64	resumed_at_size	Used with download resumption and indicates at what point the current download resumed.
public ml_file_transfer_info *	info	Points to the information object passed to the MLFileDownload method. You can access the user_data parameter through this pointer.
public asa_uint16	flags	Provides additional information. The MLFT_STATUS_FLAG_IS_BLOCKING value is set when the MLFileDownload method is blocking on a network call and the download status has not changed since the last time the observer method was called.
public asa_uint8	stop	Set to true to cancel the current download. You can resume the download in a subsequent call to the MLFileDownload method, but only if you have set the enable_resume parameter.

1.76 mlft_stream_error Structure

A structure containing status/progress information while the file upload or download is in progress.

Syntax

```
typedef struct mlft_stream_error
```

Members

All members of mlft_stream_error, including inherited members.

Variables

Modifier and Type	Variable	Description
public ss_error_code	stream_error_code	The specific stream error. For a list of possible values, see the ss_error_code enumeration in the `%SQLANY17%\SDK\Include\sserror.h` header file.
public asa_int32	system_error_code	A system-specific error code.
public char	error_string	The localized description for the system_error_code value, if available from the system, or additional information for the stream_error_code value.

1.77 MLFT_STATUS_FLAG_IS_BLOCKING Variable

Defines a bit set in the ml_file_transfer_status.flags field to indicate that the file transfer is blocked awaiting a response from the MobiLink server.

Syntax

```
#define MLFT_STATUS_FLAG_IS_BLOCKING
```

Remarks

Identical file transfer progress messages are generated periodically while this is the case.

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