UltraLite®
C- und C++-Programmierung

Version 16.0

Februar 2013

Version 16.0
Februar 2013

Inhalt

Über diese Dokumentation

Diese Dokumentation beschreibt die UltraLite C- und C++-Programmierschnittstellen. Mit UltraLite können Sie Datenbankanwendungen für Handhelds oder mobile Geräte (einschließlich iPhone und iPad) und eingebettete Geräte entwickeln.

Erforderliche Systemausstattung und unterstützte Plattformen

Entwicklungsplattformen

Wenn Sie Anwendungen mit UltraLite C++ entwickeln möchten, benötigen Sie Folgendes:

- Ein Microsoft Windows-, Linux- oder Mac-PC als Entwicklungsplattform.
- Ein unterstützter Microsoft- oder GNU C/C++-Compiler.

Zielplattformen

UltraLite C/C++ unterstützt die folgenden Zielplattformen:

- Windows Mobile 5.0 oder später
- Windows XP oder später
- Linux
- Embedded Linux
- iOS 3 und später (iPhone und iPad)
- Mac

Weitere Informationen zu unterstützten Zielplattformen finden Sie unter http://www.sybase.com/detail?id=1002288.

Anwendungsentwicklung

Dieser Abschnitt enthält Hinweise zur UltraLite-Entwicklung für C/C++-Programmierer.

Die C/C++-Schnittstellen bieten folgende Vorteile für UltraLite-Entwickler:

- Einen kleinen, leistungsfähigen Datenbankspeicher mit nativer Synchronisation.
- Die Leistungsfähigkeit, Effizienz und Flexibilität von C oder C++
- Die Möglichkeit für das Deployment von Anwendungen unter Windows Mobile, Windows PC-Plattformen, Linux Desktop, Embedded Linux, iPhone und iPad.

Alle UltraLite C/C++-Schnittstellen verwenden dieselbe UltraLite-Runtime-Engine. Die APIs bieten jeweils Zugriff auf dieselben Basisfunktionen.

Siehe auch

- „UltraLite-Datenbank erstellen“ [*UltraLite - Datenbankverwaltung*]

UltraLite C++-Anwendungsentwicklung

Schnellstart-Handbuch zur UltraLite C++-Anwendungsentwicklung

Die folgende Vorgehensweise wird üblicherweise verwendet, wenn Sie eine Anwendung mit der UltraLite C++-API entwickeln:

1. Initialisieren Sie ein ULDatabaseManager-Objekt.
2. (Optional) Aktivieren Sie die Funktionen in der UltraLite-Laufzeitumgebung.
3. Verwenden Sie eine UltraLite-Datenbank. Sie können eine Verbindung mit einer bestehenden Datenbank verwenden, eine neue erstellen, eine bestehende Datenbank löschen oder eine bestehende Datenbank validieren, um sicherzustellen, dass keine Dateien beschädigt sind.
4. Finalisieren Sie das ULDatabaseManager-Objekt.

Das ULDatabaseManager-Objekt sollte nur einmal in Ihrer Anwendung initialisiert werden und ist zu finalisieren, wenn Ihre Anwendung beendet wird. Alle Methoden der ULDatabaseManager-Klasse sind statisch. Verwenden Sie die ULError-Klasse für den Abruf von Fehlerinformationen in Ihrer UltraLite-Anwendung.

Siehe auch

- ULDatabaseManager-Klasse [UltraLite C++] auf Seite 134
- „So erstellen und Sie UltraLite C++-Anwendungen und führen ein Deployment durch.“ auf Seite 24

Hinweise zu iPhone und Mac OS X

Entwicklungsumgebung

Die Entwicklungsumgebung für iPhone und Mac OS X ist Xcode.

Build-Einstellungen

Zum Verweis auf UltraLite-Headerdateien und die Bibliothek ist die Erstellung einer benutzerdefinierten Build-Einstellung für die Kompilierung der Anwendung zu empfehlen, die auf den Standort des SQL Anywhere-Installationsverzeichnisses verweist. Setzen Sie beispielsweise *SQLANY_ROOT* auf */Anwendungen/SQLAnywhere16*. Um diese Einstellung zu erstellen, öffnen Sie den **Build**-Bereich des Projekteditors und klicken Sie auf **Add User-Defined Setting**. Geben Sie den Namen und den Wert ein.

Include-Dateien

Um die UltraLite Include-Dateien zu suchen, fügen Sie *$(SQLANY_ROOT)/sdk/include* zur Build-Einstellung **User Header Search Paths***(USER_HEADER_SEARCH_PATHS*) hinzu.

Nicht unterstützte Netzwerkprotokolloptionen für den MobiLink-Client

UltraLite für iPhone und/oder Mac OS X unterstützt folgende Netzwerkprotokolloptionen für den MobiLink-Client nicht:

- certificate_company
- certificate_unit
- client_port
- identity
- identity_password
- network_leave_open
- network_name

Verschlüsselung

Um die Ende-zu-Ende-Verschlüsselung bei der Synchronisation von Mac OS X- und iPhone-UltraLite-Clients mit einem MobiLink-Server zu verwenden, müssen Sie Ihre öffentlichen Schlüssel in einem PEM-kodierten X509-Zertifikat verpacken (im Gegensatz zu einer öffentlichen PEM-Schlüsseldatei) und einen privaten E2EE-Schlüssel angeben. Um ein PEM-kodiertes X509-Zertifikat mit einem privaten E2EE-Schlüssel zu erstellen, wird empfohlen, dass Sie das Dienstprogramm zur Zertifikatserstellung "createcert" verwenden. Nachdem Sie einen privaten E2EE-Schlüssel erhalten haben, geben Sie die Option -x an, wenn Sie den MobiLink-Server starten, und ordnen Sie den Schlüssel der e2ee_private_key-Option zu. Um die UltraLite-Clientdatenbank mit dem MobiLink-Server zu synchronisieren, führen Sie das UltraLite-Synchronisationsdienstprogramm ulsync aus und ordnen Sie den öffentlichen E2EE-Schlüssel der e2ee_public_key-Verbindungsoption zu. Das Extrahieren des öffentlichen Schlüssels aus dem Zertifikat ist erforderlich, wenn iPhone- und Nicht-iPhone-Clients gemeinsam verwendet werden. Beim Entwickeln von iPhone-UltraLite-Clients sucht das UltraLite-Synchronisationsdienstprogramm nach der Zertifikatdatei im Haupt-Ressourcen-Bundle (mainBundle) des iPhone-Entwicklungspakets, wenn die trusted_certificates- oder die e2ee_public_key-Option zugeordnet ist. Sie müssen das Zertifikat im Ressources-Ordner in Ihrem Xcode-Projekt einschließen.

Folgende Verschlüsselungsstandards werden nicht unterstützt:

- FIPS-zertifizierte Verschlüsselung

Fehlersuche bei iPhone-Anwendungen

Der Xcode-Debugger (GDB) hat Unterstützung für das schrittweise Durchgehen des Codes und Abbrechen bei *longjmp()*-Aufrufen. Anwendungen verwenden in der Regel *longjmp* nicht, aber die UltraLite-Laufzeitbibliothek verwendet es intern (manchmal, beispielsweise wenn ein Fehler signalisiert wird). Dies kann Probleme verursachen, wenn man den Anwendungscode durchgeht und UltraLite-Aufrufe überschreitet. Wenn Sie einen UltraLite-Aufruf überschreiten und eine Fehlermeldung aus dem Debugger erhalten: Starten Sie das Programm erneut, setzen Sie einen Breakpoint nach der problematischen Zeile und verwenden Sie anstelle des Überschreitens der problematische Zeile den **Continue**-Befehl. Dies hat dieselbe Wirkung, weil der Debugger am folgenden Breakpoint stoppt, vermeidet aber gleichzeitig Probleme im Zusammenhang mit *longjmp*-Aufrufen. Dies kommt wahrscheinlich vor, wenn Sie OpenConnection verwenden, um eine bestehende Datenbank zu öffnen oder zu ermitteln, ob die Datenbank nicht besteht (ein Fehler wird signalisiert, wenn die Datenbank nicht existiert).

Siehe auch

- „e2ee_public_key“ [*MobiLink - Clientadministration*]
- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]
- „Datenbanksicherheit“ [*UltraLite - Datenbankverwaltung*]
- „Dienstprogramm zum Erstellen von Zertifikaten [createcert)“ [*SQL Anywhere Server - Datenbankadministration*]
- „mlsrv16-Option -x“ [*MobiLink - Serveradministration*]
- „UltraLite-Synchronisationsdienstprogramm (ulsync)“ [*UltraLite - Datenbankverwaltung*]

UltraLite-Datenbankverbindungen

UltraLite-Anwendungen müssen eine Verbindung zur Datenbank herstellen, bevor sie ihre Daten bearbeiten können.

Die ULDatabaseManager-Klasse wird benutzt, um eine Verbindung mit einer Datenbank zu öffnen. Die ULDatabaseManager-Klasse gibt ein Nicht-Null-ULConnection-Objekt zurück, wenn eine Verbindung hergestellt wird. Verwenden Sie das ULConnection-Objekt, um die folgenden Aufgaben auszuführen:

- Festschreiben oder Zurücksetzen von Transaktionen.
- Synchronisieren von Daten mit einem MobiLink-Server.
- Zugriff auf Tabellen in der Datenbank.
- Arbeiten mit SQL-Anweisungen.
- Verarbeitung der Fehler in Ihrer Anwendung.

Achten Sie darauf, einen beschreibbaren Pfad für die Datenbankdatei anzugeben. Verwenden Sie die *NSSearchPathForDirectoriesInDomains*-Methode, um beispielsweise *NSDocumentDirectory* abzufragen.

Hinweis
Der Beispielcode befindet sich im Verzeichnis *%SQLANYSAMP16%\UltraLite\CustDB*.

Anwendungen mit mehreren Threads

Jede Verbindung und alle von ihr erstellten Objekte sollten von einem einzigen Thread verwendet werden. Wenn eine Anwendung mehrere Threads benötigt, die auf die UltraLite-Datenbank zugreifen, muss jeder Thread seine eigene Verbindung haben.

Siehe auch

- ULConnection-Klasse [UltraLite C++] auf Seite 107
- ULDatabaseManager-Klasse [UltraLite C++] auf Seite 134
- ULError-Klasse [UltraLite C++] auf Seite 148

Verbindung mit einer UltraLite-Datenbank herstellen

Verwenden Sie das ULDatabaseManager-Objekt, um eine UltraLite-Datenbank namens *sample.udb* zu erstellen oder eine Verbindung dazu herzustellen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Initialisieren Sie das ULDatabaseManager-Objekt und aktivieren Sie Funktionen in UltraLite mit dem folgenden Code:

```
if( !ULDatabaseManager::Init() ) {
    return 0;
}
ULDatabaseManager::EnableAesDBEncryption();

// Use ULDatabaseManager.Fini() when terminating the app.
```

2. Öffnen Sie mit dem folgenden Code eine Verbindung mit einer bestehenden Datenbank oder erstellen Sie eine neue Datenbank, wenn die angegebene Datenbankdatei nicht vorhanden ist:

```
ULConnection * conn;
ULError ulerr;

conn =
ULDatabaseManager::OpenConnection( "dbf=sample.udb;dbkey=aBcD1234",
&ulerr );
if( conn == NULL ) {
    if( ulerr.GetSQLCode() == SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
        conn =
ULDatabaseManager::CreateDatabase( "dbf=sample.udb;dbkey=aBcD1234",
&ulerr );
        if( conn == NULL ) {
            // write code that uses ulerr to determine what happened
            return 0;
        }
        // add code to create the schema for your database
```

```
        } else {
            // write code that uses ulerr to determine what happened
            return 0;
        }
    }
    assert( conn != NULL );
```

In diesem Schritt deklarieren Sie ein ULError-Objekt, das Fehlerinformationen für den Fall enthält, dass die Verbindung nicht erfolgreich verlaufen ist.

Ergebnisse

Eine Verbindung zur *sample.udb*-Datenbank wird hergestellt.

Datenerstellung und -änderung mit SQL-Anweisungen

UltraLite-Anwendungen können auf Tabellendaten durch das Ausführen von SQL-Anweisungen oder mit der ULTable-Klasse zugreifen. In diesem Abschnitt wird der Datenzugriff mit SQL-Anweisungen beschrieben.

In diesem Abschnitt wird erläutert, wie die folgenden Aufgaben mit SQL erledigt werden:

- Zeilen einfügen, löschen und aktualisieren
- Zeilen in eine Ergebnismenge auslesen
- Durch die Zeilen einer Ergebnismenge blättern

In diesem Abschnitt wird die SQL-Sprache nicht behandelt.

Siehe auch

- „UltraLite-SQL-Anweisungen“ [*UltraLite - Datenbankverwaltung*]
- „Datenerstellung und -änderung mit der ULTable-Klasse“ auf Seite 15

Datenmodifikation mit INSERT, UPDATE und DELETE

In UltraLite können Sie die SQL-Datenmanipulation mit der ExecuteStatement-Methode (Mitglied der ULPreparedStatement-Klasse) durchführen.

Siehe auch

- ULPreparedStatement-Klasse [UltraLite C++] auf Seite 156

Eine Zeile in eine Tabelle einfügen

UltraLite kennzeichnet Abfrageparameter mit dem ?-Zeichen. Für eine INSERT, UPDATE oder DELETE-Anweisung wird jedes ? entsprechend seiner Stellung in der vorbereiteten Anweisung referenziert. Das erste ? wird z.B. als Parameter 1 referenziert, das zweite als Parameter 2.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Deklarieren Sie ein ULPreparedStatement mit dem folgenden Code:

   ```
   ULPreparedStatement * prepStmt;
   ```

2. Bereiten Sie eine SQL-Anweisung für die Ausführung vor.

   Der folgende Code bereitet eine INSERT-Anweisung für die Ausführung vor:

   ```
   prepStmt = conn->PrepareStatement("INSERT INTO MyTable(MyColumn1) VALUES
   (?)");
   ```

3. Prüfen Sie bei der Vorbereitung der Anweisung, ob Fehler auftreten.

   Beispiel: Der folgende Code ist nützlich, wenn Sie eine Prüfung auf SQL-Syntaxfehler vornehmen:

   ```
   if( prepStmt == NULL ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       return;
   }
   ```

4. Legen Sie Werte zum Ersetzen der ?- Zeichen in der vorbereiteten Anweisung fest.

   Der folgende Code legt das ?- Zeichen während der Überprüfung auf Fehler mit "some value" fest. Beispiel: Ein Fehler wird abgefangen, wenn die Parameterordinalzahl außerhalb des zulässigen Bereichs für die Anzahl der Parameter in der vorbereiteten Anweisung liegt.

   ```
   if( !prepStmt->SetParameterString(1, "some value") ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       return;
   }
   ```

5. Führen Sie die vorbereitete Anweisung aus und fügen Sie Daten in die Datenbank ein.

   Der nachstehende Code prüft, ob Fehler vorhanden sind, die nach dem Ausführen der Anweisung auftreten könnten. Beispiel: Ein Fehler wird zurückgegeben, wenn ein doppelter Indexwert in einem eindeutigen Index gefunden wird.

   ```
   bool success;
   success = prepStmt->ExecuteStatement();
   if( !success ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
   } else {
       // Use the following line if you are interested in the number of rows
   inserted ...
       ul_u_long rowsInserted = prepStmt->GetRowsAffectedCount();
   }
   ```

6. Bereinigen Sie die Ressourcen der vorbereiteten Anweisung.

   Der folgende Code gibt die Ressourcen frei, die vom Objekt der vorbereiteten Anweisung verwendet werden. Auf dieses Objekt darf nicht mehr zugegriffen werden, nachdem die Close-Methode aufgerufen wird.

   ```
   prepStmt->Close();
   ```

7. Schreiben Sie die Daten in der Datenbank fest.

   Der folgende Code speichert die Daten in der Datenbank und verhindert Datenverlust. Die Daten aus Schritt 5 gehen verloren, wenn die Geräteanwendung unerwartet abbricht, bevor die Anwendung einen Commit-Aufruf abschließen kann.

   ```
   conn->Commit();
   ```

Ergebnisse

Eine neue Zeile wird zu MyTable hinzugefügt, wobei der MyColumn1-Wert auf eine Zeichenfolge "some value" gesetzt wird.

Einer Zeile in einer Tabelle löschen

UltraLite kennzeichnet Abfrageparameter mit dem ?-Zeichen. Für eine INSERT, UPDATE oder DELETE-Anweisung wird jedes ? entsprechend seiner Stellung in der vorbereiteten Anweisung referenziert. Das erste ? wird z.B. als Parameter 1 referenziert, das zweite als Parameter 2.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Deklarieren Sie ein ULPreparedStatement mit dem folgenden Code:

   ```
   ULPreparedStatement * prepStmt;
   ```

2. Bereiten Sie eine SQL-Anweisung für die Ausführung vor.

   Der folgende Code bereitet eine DELETE-Anweisung für die Ausführung vor:

   ```
   prepStmt = conn->PrepareStatement("DELETE FROM MyTable(MyColumn1) VALUES
   (?)");
   ```

3. Prüfen Sie bei der Vorbereitung der Anweisung, ob Fehler auftreten.

   Beispiel: Der folgende Code ist nützlich, wenn Sie eine Prüfung auf SQL-Syntaxfehler vornehmen:

   ```
   if( prepStmt == NULL ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       return;
   }
   ```

4. Legen Sie Werte zum Ersetzen der ?- Zeichen in der vorbereiteten Anweisung fest.

   Der folgende Code legt das ?- Zeichen während der Überprüfung auf Fehler mit 7 fest. Beispiel: Ein Fehler wird abgefangen, wenn die Parameterordinalzahl außerhalb des zulässigen Bereichs für die Anzahl der Parameter in der vorbereiteten Anweisung liegt.

   ```
   ul_s_long value_to_delete = 7;
   if( !prepStmt->SetParameterInt(1, value_to_delete) ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error.
       return;
   }
   ```

5. Führen Sie die vorbereitete Anweisung aus und löschen Sie die Daten aus der Datenbank.

   Der nachstehende Code prüft, ob Fehler vorhanden sind, die nach dem Ausführen der Anweisung auftreten könnten. Zum Beispiel wird ein Fehler zurückgegeben, wenn Sie versuchen, eine Zeile zu löschen, auf die ein Fremdschlüssel verweist.

   ```
   bool success;
   success = prepStmt->ExecuteStatement();
   if( !success ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
   } else {
       // Use the following line if you are interested in the number of rows
   deleted ...
       ul_u_long rowsDeleted = prepStmt->GetRowsAffectedCount();
   }
   ```

6. Bereinigen Sie die Ressourcen der vorbereiteten Anweisung.

   Der folgende Code gibt die Ressourcen frei, die vom Objekt der vorbereiteten Anweisung verwendet werden. Auf dieses Objekt darf nicht mehr zugegriffen werden, nachdem die Close-Methode aufgerufen wird.

   ```
   prepStmt->Close();
   ```

7. Schreiben Sie die Daten in der Datenbank fest.

   Der folgende Code speichert die Daten in der Datenbank und verhindert Datenverlust. Die Daten aus Schritt 5 gehen verloren, wenn die Geräteanwendung unerwartet abbricht, bevor die Anwendung einen Commit-Aufruf abschließen kann.

   ```
   conn->Commit();
   ```

Ergebnisse

Zeileneinträge werden aus MyTable gelöscht, wenn der MyColumn-Wert in der Tabelle gleich 7 ist.

Eine Zeile in einer Tabelle aktualisieren

UltraLite kennzeichnet Abfrageparameter mit dem ?-Zeichen. Für eine INSERT, UPDATE oder DELETE-Anweisung wird jedes ? entsprechend seiner Stellung in der vorbereiteten Anweisung referenziert. Das erste ? wird z.B. als Parameter 1 referenziert, das zweite als Parameter 2.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Deklarieren Sie ein ULPreparedStatement mit dem folgenden Code:

   ```
   ULPreparedStatement * prepStmt;
   ```

2. Bereiten Sie eine SQL-Anweisung für die Ausführung vor.

   Der folgende Code bereitet eine UPDATE-Anweisung für die Ausführung vor:

   ```
   prepStmt = conn->PrepareStatement("UPDATE MyTable SET MyColumn = ? WHERE
   MyColumn = ?");
   ```

3. Prüfen Sie bei der Vorbereitung der Anweisung, ob Fehler auftreten.

   Beispiel: Der folgende Code ist nützlich, wenn Sie eine Prüfung auf SQL-Syntaxfehler vornehmen:

   ```
   if( prepStmt == NULL ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       return;
   }
   ```

4. Legen Sie Werte zum Ersetzen der ?- Zeichen in der vorbereiteten Anweisung fest.

   Der folgende Code legt das ?- Zeichen während der Überprüfung auf Fehler auf Ganzzahlwerte fest. Beispiel: Ein Fehler wird abgefangen, wenn die Parameterordinalzahl außerhalb des zulässigen Bereichs für die Anzahl der Parameter in der vorbereiteten Anweisung liegt.

   ```
   bool success;
   success = prepStmt->SetParameterInt( 1, 25 );
   if( success ) {
       success = prepStmt->SetParameterInt( 2, -1 );
   }
   if( !success ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       return;
   }
   ```

5. Führen Sie die vorbereitete Anweisung aus und aktualisieren Sie die Daten in der Datenbank.

   Der nachstehende Code prüft, ob Fehler vorhanden sind, die nach dem Ausführen der Anweisung auftreten könnten. Beispiel: Ein Fehler wird zurückgegeben, wenn ein doppelter Indexwert in einem eindeutigen Index gefunden wird.


```
success = prepStmt->ExecuteStatement();
if( !success ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
} else {
    // if you are interested in the number of rows updated ...
    ul_u_long rowsUpdated = prepStmt->GetRowsAffectedCount();
}
```

6. Bereinigen Sie die Ressourcen der vorbereiteten Anweisung.

   Der folgende Code gibt die Ressourcen frei, die vom Objekt der vorbereiteten Anweisung verwendet werden. Auf dieses Objekt darf nicht mehr zugegriffen werden, nachdem die Close-Methode aufgerufen wird.

   ```
   prepStmt->Close();
   ```

7. Schreiben Sie die Daten in der Datenbank fest.

   Der folgende Code speichert die Daten in der Datenbank und verhindert Datenverlust. Die Daten aus Schritt 5 gehen verloren, wenn die Geräteanwendung unerwartet abbricht, bevor die Anwendung einen Commit-Aufruf abschließen kann.

   ```
   conn->Commit();
   ```

Ergebnisse

In diesem Szenario werden Zeileneinträge in MyTable aktualisiert, wenn der MyColumn-Wert gleich -1 ist. Der Wert wird auf 25 aktualisiert.

Daten mit SELECT abrufen

Führen Sie eine SELECT-Anweisung aus, um Daten aus einer UltraLite-Datenbank abzurufen, und bearbeiten sie die zurückgegebene Ergebnismenge.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgaben

1. Deklarieren Sie die erforderlichen Variablen mit dem folgenden Code:

   ```
   ULPreparedStatement * prepStmt;
   ULResultSet * resultSet;
   ```

2. Bereiten Sie eine SQL-Anweisung für die Ausführung vor.

   Der folgende Code bereitet eine SELECT-Anweisung für die Ausführung vor:

   ```
   prepStmt = conn->PrepareStatement("SELECT MyColumn1 FROM MyTable");
   ```

3. Prüfen Sie bei der Vorbereitung der Anweisung, ob Fehler auftreten.

   Beispiel: Der folgende Code ist nützlich, wenn Sie eine Prüfung auf SQL-Syntaxfehler vornehmen:

   ```
   if( prepStmt == NULL ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       return;
   }
   ```

4. Führen Sie die SQL-Anweisung aus und geben Sie ein Ergebnismengenobjekt zurück, das verwendet werden kann, um auf die Ergebnisse der Abfrage zuzugreifen.

   ```
   resultSet = prepStmt->ExecuteQuery();
   if( resultSet == NULL ) {
       const ULError * ulerr;
       ulerr = conn->GetLastError();
       // write code to handle the error
       prepStmt->Close();
       return;
   }
   ```

5. Durchlaufen Sie die Zeilen mithilfe der Next-Methode. Speichern Sie das Ergebnis als Zeichenfolge in einem Puffer.

   Die Next-Methode bewegt den Cursor in die nächste Zeile der Ergebnismenge. Das ULResultSet-Objekt ist auf einer Zeile positioniert, wenn der Aufruf TRUE zurückgibt. Andernfalls gilt: Wenn der Aufruf FALSE zurückgibt, wurden alle Zeilen durchlaufen.

   ```
   while( resultSet->Next() ) {
       char buffer[ 100 ];
       resultSet->GetString( 1, buffer, 100 );
       printf( "MyColumn = %s\n", buffer );
   }
   ```

6. Bereinigen Sie die Ressourcen der vorbereiteten Anweisung und des Ergebnismengeobjekts.

   Auf das Objekt der vorbereiteten Anweisung darf nicht zugegriffen werden, nachdem die Close-Methode aufgerufen wurde.

   ```
   resultSet->Close();
   prepStmt->Close();
   ```

Ergebnisse

Das Ergebnis der SELECT-Anweisung enthält eine Zeichenfolge, die dann auf der Eingabeaufforderung ausgegeben wird.

Siehe auch

- ULPreparedStatement.ExecuteQuery-Methode [UltraLite C++] auf Seite 159

Erstellen und Abrufen von Schemabeschreibungen

Mit der GetResultSetSchema-Methode können Sie Schemainformationen über eine Ergebnismenge abfragen, wie z.B. Spaltennamen, die Gesamtzahl der Spalten, Spalten-Gesamtstellen, Spaltengröße sowie Spalten-SQL-Typen.

Beispiel

Das folgende Beispiel zeigt, wie Sie mit der GetResultSetSchema-Methode Schemainformationen in einer Eingabeaufforderung anzeigen:

```
const char * name;
int column_count;
const ULResultSetSchema & rss = prepStmt->GetResultSetSchema();
int column_count = rss.GetColumnCount();
for( int i = 1; i < column_count; i++ ) {
    name = rss.GetColumnName( i );
    printf( "id = %d, name = %s\n", i, name );
}
```

In diesem Beispiel werden die erforderlichen Variablen deklariert und das ULResultSetSchema-Objekt wird zugewiesen. Sie können ein ULResultSetSchema-Objekt aus der Ergebnismenge selbst erhalten, aber dieses Beispiel zeigt, wie das Schema verfügbar ist, nachdem die Anweisung vorbereitet wurde und bevor die Abfrage ausgeführt wird. Die Anzahl der Zeilen in der Ergebnismenge werden gezählt und der Name jeder einzelnen Spalte wird angezeigt.

Siehe auch

- ULPreparedStatement.GetResultSetSchema-Methode [UltraLite C++] auf Seite 161

Navigation in SQL-Ergebnismengen

Sie können durch eine Ergebnismenge navigieren, indem Sie die mit der ULResultSet-Objekt verbundenen Methoden verwenden.

Die Ergebnismengen-Klasse bietet Ihnen die folgenden Methoden zur Navigation in einer Ergebnismenge:

- **AfterLast** Bewegt den Cursor hinter die letzte Zeile.
- **BeforeFirst** Bewegt den Cursor vor die erste Zeile.
- **First** Bewegt den Cursor in die erste Zeile.
- **Last** Bewegt den Cursor in die letzte Zeile.
- **Next** Bewegt den Cursor in die nächste Zeile.
- **Previous** Bewegt den Cursor in die vorige Zeile.
- **Relative(offset)** Bewegt den Cursor um eine bestimmte Anzahl von Zeilen, der im Offsetwert angegeben ist, relativ zur aktuellen Zeile. Positive Offsetwerte bewegen den Cursor in der Ergebnismenge vorwärts, relativ zur aktuellen Position des Cursors in der Ergebnismenge. Negative

Offsetwerte bewegen ihn rückwärts in der Ergebnismenge. Der Offsetwert Null ändert die Cursorposition nicht, ermöglicht aber das Einlesen von Daten in den Zeilenpuffer.

Siehe auch

- ULResultSet-Klasse [UltraLite C++] auf Seite 166

Datenerstellung und -änderung mit der ULTable-Klasse

UltraLite-Anwendungen können auf Tabellendaten durch das Ausführen von SQL-Anweisungen oder mit der ULTable-Klasse zugreifen. In diesem Abschnitt wird der Datenzugriff mit der ULTable-Klasse beschrieben.

In diesem Abschnitt wird erläutert, wie die folgenden Aufgaben mit der ULTable-Klasse erledigt werden:

- Durch die Zeilen einer Tabelle blättern
- Auf die Werte der aktuellen Zeile zugreifen
- Zeilen in einer Tabelle mit den Find- und Lookup-Methoden ausfindig machen
- Zeilen einfügen, löschen und aktualisieren

> **Vorsicht**
> Aktualisieren Sie nicht den Primärschlüssel einer Zeile. Löschen Sie stattdessen die Zeile und fügen Sie eine neue Zeile hinzu.

Siehe auch

- „Datenerstellung und -änderung mit SQL-Anweisungen“ auf Seite 7

Zeilennavigation

Die UltraLite C++-API bietet Ihnen mehrere Methoden zur Navigation durch eine Tabelle, um diverse Navigationsaufgaben durchzuführen.

Das ULTable-Objekt bietet Ihnen die folgenden Methoden zur Navigation durch eine Tabelle:

- **AfterLast** Bewegt den Cursor hinter die letzte Zeile.
- **BeforeFirst** Bewegt den Cursor vor die erste Zeile.
- **First** Bewegt den Cursor in die erste Zeile.
- **Last** Bewegt den Cursor in die letzte Zeile.
- **Next** Bewegt den Cursor in die nächste Zeile.
- **Previous** Bewegt den Cursor in die vorige Zeile.

- **Relative(offset)** Bewegt den Cursor um eine bestimmte Anzahl von Zeilen, der im Offsetwert angegeben ist, relativ zur aktuellen Zeile. Positive Offsetwerte bewegen den Cursor in der Ergebnismenge vorwärts, relativ zur aktuellen Position des Cursors in der Ergebnismenge. Negative Offsetwerte bewegen ihn rückwärts in der Ergebnismenge. Der Offsetwert Null ändert die Cursorposition nicht, ermöglicht aber das Einlesen von Daten in den Zeilenpuffer.

Siehe auch

- ULTable-Klasse [UltraLite C++] auf Seite 207

Beispiel

Der Code im folgenden Beispiel öffnet die Tabelle MyTable und zeigt den Wert der Spalte MyColumn für jede Zeile an:

```
char buffer[ 100 ];
ul_column_num column_id;
ULTable * tbl = conn->OpenTable( "MyTable" );
if( tbl == NULL ) {
    const ULError * ulerr;
    ulerr = conn->GetLastError();
    // write code to handle the error
    return;
}
column_id = tbl->GetTableSchema().GetColumnID( "MyColumn" );
if( column_id == 0 ) {
    // the column "MyColumn" likely does not exist.  Handle the error.
    tbl->Close();
    return;
}
while( tbl->Next() ) {
    tbl->GetString( column_id, buffer, 100 );
    printf( "%s\n", buffer );
}
tbl->Close();
```

Die Zeilen der Tabelle werden der Anwendung vorgelegt, wenn Sie das ULTable-Objekt öffnen. Standardmäßig werden die Zeilen in der Reihenfolge des Primärschlüsselwertes sortiert, Sie können jedoch einen Index angeben und eine Tabelle öffnen, um in einer bestimmten Reihenfolge auf die Zeilen zuzugreifen.

Beispiel

Mit dem Code im folgenden Beispiel wird zur ersten Zeile der Tabelle MyTable gewechselt, wie es der Index ix_col vorsieht:

```
ULTable * tbl = conn->OpenTable( "MyTable", "ix_col" );
```

UltraLite-Modi

Der UltraLite-Modus legt fest, wie Werte im Puffer verwendet werden. Sie können einen der folgenden UltraLite-Modi festlegen:

- **Einfügungsmodus (insert)** Daten im Puffer werden der Tabelle als neue Zeile hinzugefügt, wenn die insert-Methode aufgerufen wird.

- **Aktualisierungsmodus (update)** Daten im Puffer ersetzen die aktuelle Zeile, wenn die update-Methode aufgerufen wird.

- **Suchmodus (find)** Dieser Modus sucht eine Zeile, deren Wert genau zu den Daten im Puffer passt, wenn eine der find-Methoden aufgerufen wird.

- **Nachschlagemodus (lookup)** Sucht eine Zeile, deren Wert genau zu den Daten im Puffer passt oder größer ist, wenn einer der Nachschlagemodi aufgerufen wird.

Der Modus wird festgelegt, indem die betreffende Methode aufgerufen wird. Z.B: InsertBegin, UpdateBegin, FindBegin usw.

Zeileneinfügungen

Die Schritte zum Einfügen einer Zeile sind den Schritten zum Aktualisieren von Zeilen sehr ähnlich, außer dass es nicht erforderlich ist, eine Zeile in der Tabelle ausfindig zu machen, bevor der Einfügevorgang vorgenommen werden kann.

Wenn Sie keinen Wert für eine der Spalten festlegen und diese Spalte einen Standardwert enthält, wird der Standardwert verwendet. Wenn die Spalte keinen Standardwert enthält, wird einer der folgenden Einträge benutzt:

- Bei nullwertfähigen Spalten: NULL
- Bei nummerischen Spalten, die NULL nicht zulassen: Null (0).
- Bei Zeichenspalten, die NULL nicht zulassen: eine leere Zeichenfolge
- Um explizit einen Wert auf NULL zu setzen, verwenden Sie die SetNull-Methode.

Beispiel

Mit dem folgenden Code wird gezeigt, wie eine neue Zeile eingefügt wird:

```
ULTable * tbl = conn->OpenTable("MyTable");
bool success;
tbl->InsertBegin();  // enter "Insert mode"
tbl->SetInt("id", 3);
tbl->SetString("lname", "Smith");
tbl->SetString("fname", "Mary");
success = tbl->Insert();
conn->Commit();
tbl->Close();
```

In diesem Beispiel ist die tbl-Variable so festgelegt, dass sie MyTable öffnet. Die Werte für die einzelnen Spalten werden im aktuellen Zeilenpuffer festgelegt. Spalten können mit Namen oder ID referenziert werden. Die Insert-Methode veranlasst, dass die temporäre Zeilenpufferwerte in die Datenbank eingefügt werden. Die Ergebnisse werden dann festgeschrieben und angezeigt. Ressourcen werden mit der Close-Methode freigegeben.

Zeilen aktualisieren

Verwenden Sie die Update-Methode, um eine Zeile in einer Tabelle zu aktualisieren.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Wechseln Sie zu der zu aktualisierenden Zeile.

   Sie können dafür entweder die Tabelle abrollen oder die find- und lookup-Methoden verwenden.

2. Aktivieren Sie den Update-Modus.

   Die folgende Anweisung startet den Aktualisierungsmodus für die Tabelle tbl.

   ```
   tbl->UpdateBegin();
   ```

3. Legen Sie die neuen Werte für die zu aktualisierende Zeile fest. Die nachstehende Anweisung setzt die id-Spalte im Puffer auf 3.

   ```
   tbl->SetInt("id", 3);
   ```

4. Führen Sie die Aktualisierung aus.

   ```
   tbl->Update();
   ```

Vorsicht
Wenn Sie die Find- und Update-Methoden verwenden, ist Ihr Zeiger möglicherweise nicht an der erwarteten Position, nachdem eine Spalte aktualisiert wurde, die im Suchkriterium enthalten ist. In einigen Fällen ist es empfehlenswert, eine SQL-Anweisung zu verwenden, wenn mehrere Zeilen aktualisiert werden.

Ergebnisse

Nach der Aktualisierung ist die aktualisierte Zeile die aktuelle Zeile.

Nächste Schritte

Führen Sie zusätzliche SQL-Vorgänge in der Datenbank aus und verwenden Sie dann die Commit-Methode, um Änderungen in der Datenbank festzuschreiben. ausgeführt.

Siehe auch

- „Transaktionsverwaltung“ auf Seite 21

Find- und Lookup-Modi zum Suchen nach Zeilen

UltraLite verfügt über verschiedene Modi für die Arbeit mit Daten. Zwei dieser Methoden, nämlich den Find- und den Lookup-Modus, können Sie für die Suche verwenden. Das ULTable-Objekt verfügt über zwei Methoden, die diesen Modi zum Ausfindigmachen bestimmter Zeilen in einer Tabelle entsprechen:

Hinweis
Die Spalten, die mit der Find- und der Lookup-Methode gesucht werden, müssen sich in dem Index befinden, der zum Öffnen der Tabelle benutzt wird.

- **Find-Methoden** Diese Methoden platzieren den Cursor auf die erste Zeile, die den angegebenen Suchwerten genau entspricht, und zwar anhand der Sortierreihenfolge, die beim Öffnen des ULTable-Objekts angegeben wurde. Sollte die Suche nicht erfolgreich sein, wird die Anwendung vor der ersten oder nach der letzten Zeile positioniert.
- **Lookup-Methoden** Sie wechseln gemäß der Sortierfolge, die beim Öffnen des ULTable-Objekts angegeben wurde, zur ersten Zeile, die größer gleich dem angegebenen Suchwert ist.

Beispiel

In diesem Beispiel wird eine Tabelle namens MyTable verwendet, die mit den folgenden SQL-Anweisungen erstellt wurden:

```
CREATE TABLE MyTable( id int primary key, lname char(100), fname char(100) )
CREATE INDEX ix_lname ON MyTable ( lname )
```

Der folgende Code zeigt alle fname-Spalteninhalte, in denen die lname-Spalte "Smith" ist:

```
ULTable * tbl = conn->OpenTable( "MyTable", "ix_lname" );
char buffer[ 100 ];
bool found;
tbl->FindBegin();  // enter "Find mode"
tbl->SetString( "lname", "Smith" );  // set pointer row buffer to "Smith"
found = tbl->FindFirst();
while( found ) {
    tbl->GetString( 3, buffer, 100 );
    printf( "%s\n", buffer );
    found = tbl->FindNext();
}
tbl->Close();
```

In diesem Beispiel wird die tbl-Variable so festgelegt, dass sie MyTable mit dem ix_lname-Index öffnet, damit Zeilen in derselben Reihenfolge zurückgegeben werden wie die lname-Spalte. ULTable-Objekte verwenden die Werte im Zeilenpuffer, wenn sie eine Suche durchführen. Dieser Puffer wird als "Smith" festgelegt, wie in der SetString-Methode definiert. Die FindFirst-Methode zeigt, dass der Durchlauf bei der ersten Zeile beginnen soll, deren lname auf "Smith" festgelegt ist. Der Zeiger wird hinter die letzte Zeile der Tabelle positioniert, wenn keine Zeilen vorhanden sind, deren lname auf "Smith" festgelegt sind. Die Spalte fname wird von der GetString-Methode abgerufen, weil die Spalte fname eine Spalten-ID 3 hat. Die Ergebnisse werden dann angezeigt und die Ressourcen werden freigegeben.

Siehe auch

- ULTable-Klasse [UltraLite C++] auf Seite 207

Zugriff auf Werte der aktuellen Zeile

Ein ULTable-Objekt befindet sich immer an einer der folgenden Positionen:

- Vor der ersten Zeile der Tabelle.
- In einer Zeile der Tabelle.
- Nach der letzten Zeile der Tabelle.

Wenn sich das ULTable-Objekt in einer Zeile befindet, können Sie je nach Datentyp eine von mehreren Methoden verwenden, um die Werte der Spalten in dieser Zeile abzurufen oder zu ändern.

Spaltenwerte abrufen

Das ULTable-Objekt bietet eine Reihe von Methoden für den Abruf von Spaltenwerten. Diese Methoden haben den Spaltennamen oder die Spalten-ID als Argument.

Im folgenden Beispiel werden zwei Arten gezeigt, um einen "age"-Wert aus einer offenen Tabelle zu beziehen, wobei angenommen wird, dass "age" die erste Spalte der Tabelle ist:

```
ul_s_long age1 = tbl->GetInt( 1 );
ul_s_long age2 = tbl->GetInt( "age" );
assert( age1 == age2 );
```

Die Verwendung der Spalten-ID-Version des Wertabrufs hat Performancevorteile, wenn die Werte in einer Schleife abgerufen werden.

Spaltenwerte ändern

Zusätzlich zu den Methoden zum Abrufen von Werten gibt es Methoden, die Werte festlegen. Diese Methoden nehmen den Spaltennamen oder die Spalten-ID und den Wert als Argument.

Der folgende Code zeigt beispielsweise zwei Arten, wie ein Zeichenfolgenwert für eine Zeile mit den Zeichenfolgenspalten lname und fname festgelegt wird, wobei angenommen wird, dass lname die erste Spalte in der Tabelle ist.

```
tbl->SetString( 1, last_name );
tbl->SetString( "fname", first_name );
```

Durch das Festlegen von Spaltenwerten ändern Sie die Daten in der Datenbank nicht direkt. Sie können den Spalten auch dann Werte zuordnen, wenn Sie sich vor der ersten Zeile oder hinter der letzten Zeile der Tabelle befinden. Versuchen Sie nicht, auf Daten zuzugreifen, wenn die aktuelle Zeile nicht festgelegt ist. Beispiel: Der Versuch, den Spaltenwert abzufragen, ist im folgenden Beispiel falsch:

```
// This code is incorrect
tbl->BeforeFirst();
tbl = tbl.GetInt( cust_id );
```

Casting von Werten

Die von Ihnen gewählte Methode muss dem Datentyp entsprechen, den Sie zuordnen möchten. UltraLite wandelt automatisch Datentypen der Datenbank um, wenn sie kompatibel sind, und daher können Sie die GetString-Methode verwenden, um einen Ganzzahlwert in eine Zeichenfolge-Variable zu übergeben usw.

Siehe auch

- „CAST-Funktion [Datentypkonvertierung]“ [*UltraLite - Datenbankverwaltung*]
- „CONVERT-Funktion [Datentypkonvertierung]“ [*UltraLite - Datenbankverwaltung*]

Zeilenlöschungen

Die Schritte zum Löschen einer Zeile sind einfacher als zum Einfügen oder Aktualisieren von Zeilen. Es gibt keinen Löschmodus, der dem Einfügungs- oder Aktualisierungsmodus entsprechen würde.

Sie löschen eine Zeile, indem Sie den Cursor auf die Zeile bewegen, die Sie löschen wollen, und dann die ULTable.Delete-Methode verwenden.

Siehe auch

- ULResultSet.Delete-Methode [UltraLite C++] auf Seite 171

Beispiel

Der folgende Code veranschaulicht, wie die erste Zeile in der Tabelle gelöscht wird:

```
tbl->First();
tbl->Delete();
```

Transaktionsverwaltung

Transaktionen werden implizit durch die erste Anweisung gestartet, um die Datenbank zu ändern, und müssen explizit festgeschrieben oder zurückgesetzt werden.

Wenn Sie eine Transaktion festschreiben möchten, verwenden Sie die ULConnection.Commit-Methode.

Um eine Transaktion zurückzusetzen, verwenden Sie die ULConnection.Rollback-Methode.

Siehe auch

- ULConnection.Commit-Methode [UltraLite C++] auf Seite 112
- ULConnection.Rollback-Methode [UltraLite C++] auf Seite 127
- „UltraLite-Transaktionsverarbeitung“ [*UltraLite - Datenbankverwaltung*]

Zugriff auf Schemainformationen

Sie können über den Programmcode Ergebnismengen- oder Datenbankstrukturbeschreibungen abrufen. Diese Beschreibungen sind als Schemainformationen bekannt und diese Informationen sind über die UltraLite C API-Schemaklassen verfügbar.

> **Hinweis**
> Sie können das Schema nicht mit der UltraLite C-API ändern. Sie können nur die Schemainformationen abrufen.

Sie können auf die folgenden Schemaobjekte und Informationen zugreifen:

- **ULResultSetSchema** Beschreibt eine Abfrage oder Daten in einer Tabelle. Es gibt Bezeichner, Name und Typinformationen der einzelnen Spalten und die Anzahl der Spalten in der Tabelle an. ULResultSetSchema-Klassen können aus den folgenden Klassen abgerufen werden:
  - ULPreparedStatement
  - ULResultSet
  - ULTable
- **ULDatabaseSchema** Gibt die Anzahl und Namen der Tabellen und Publikationen in der Datenbank sowie die globalen Eigenschaften wie Datums- und Zeitformat an. ULDatabaseSchema-Klassen können von ULConnection-Klassen abgerufen werden.
- **ULTableSchema** Zeigt Informationen über die Spalten- und Indexkonfigurationen an. Die Spalteninformationen in der ULTableSchema-Klasse ergänzen die verfügbaren Informationen aus der ULResultSetSchema-Klasse. Zum Beispiel können Sie ermitteln, ob Spalten Standardwerte haben oder Nullwerte zulassen. ULTableSchema-Klassen können von ULTable-Klassen abgerufen werden.
- **ULIndexSchema** Gibt Informationen über die Spalten im Index zurück. ULIndexSchema-Klassen können aus ULTableSchema-Klassen abgerufen werden.

Die ULResultSetSchema-Klasse wird als Konstantenreferenz zurückgegeben, anders als die ULDatabaseSchema-, ULTableSchema- und ULIndexSchema-Klassen, die als Zeiger zurückgegeben werden. Sie können keine Klasse schließen, die eine Konstantenreferenz zurückgibt, aber Sie müssen Klassen schließen, die als Zeiger zurückgegeben werden.

Mit dem folgenden Code wird die richtige und falsche Verwendung des Schließens von Schemaklassen gezeigt:

```
// This code demonstrates proper use of the ULResultSetSchema class:
const ULResultSetSchema & rss = prepStmt->GetResultSetSchema();
c_count = prepStmt->GetSchema().GetColumnCount();

// This code demonstrates proper use of the ULDatabaseSchema class:
ULDatabaseSchema * dbs = conn->GetResultSetSchema();
t_count = dbs->GetTableCount();
dbs->Close();  // This line is required.

// This code demonstrates improper use of the ULDatabaseSchema class
// because the object needs to be closed using the Close method:
t_count = conn->GetResultSetSchema()->GetTableCount();
```

Siehe auch

- ULPreparedStatement-Klasse [UltraLite C++] auf Seite 156
- ULResultSet-Klasse [UltraLite C++] auf Seite 166
- ULTable-Klasse [UltraLite C++] auf Seite 207
- ULConnection-Klasse [UltraLite C++] auf Seite 107

Fehlerbehandlung

Die UltraLite C++-API enthält ein ULError-Objekt, das zum Abrufen von Fehlerinformationen benutzt werden soll. Einige Methoden der API geben einen booleschen Wert zurück und zeigen an, ob der

Methodenaufruf erfolgreich war. In einigen Fällen wird NULL zurückgegeben, wenn ein Fehler auftritt. Das ULConnection-Objekt enthält eine GetLastError-Methode, die ein ULError-Objekt zurückgibt.

Verwenden Sie den SQLCode, um einen Fehler zu diagnostizieren. Zusätzlich zum SQLCode können Sie die GetParameterCount- und GetParameter-Methoden verwenden, um zu ermitteln, ob zusätzliche Parameter existieren, mit denen zusätzliche Informationen über den Fehler bereitgestellt werden können.

Zusätzlich zur expliziten Fehlerbehandlung unterstützt UltraLite eine Fehler-Callback-Funktion. Wenn Sie eine Callback-Funktion registrieren, ruft UltraLite die Funktion jedes Mal auf, wenn ein UltraLite-Fehler auftritt. Die Callback-Funktion steuert den Verarbeitungsfluss der Anwendung nicht, doch sie gestattet es Ihnen, über alle Fehler informiert zu werden. Die Verwendung einer Callback-Funktion ist besonders während der Anwendungsentwicklung und der Fehlersuche hilfreich.

Siehe auch

- „Praktische Einführung: Erstellen einer Windows-Anwendung mit der C++-API“ auf Seite 75
- ULSetErrorCallback-Methode [UltraLite Embedded SQL] auf Seite 256
- „SQL Anywhere-Fehlermeldungen - sortiert nach Sybase-Fehlercode“ [*Fehlermeldungen*]

MobiLink-Datensynchronisation

UltraLite-Anwendungen können Daten mit einer zentralen Datenbank synchronisieren. Die Synchronisation erfordert die MobiLink-Synchronisationssoftware, die in SQL Anywhere enthalten ist.

Die UltraLite C++-API unterstützt die TCP/IP-, TLS-, HTTP- und HTTPS-Synchronisation. Die Synchronisation wird durch die UltraLite-Anwendung eingeleitet. Die Methoden und Eigenschaften des connection-Objekts können zur Steuerung der Synchronisation verwendet werden.

Siehe auch

- „UltraLite-Clients“ [*UltraLite - Datenbankverwaltung*]
- ul_sync_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 96
- „UltraLite-Synchronisationsparameter“ [*UltraLite - Datenbankverwaltung*]

Schließen der UltraLite-Datenbankverbindung

Geben Sie Software-Ressourcen frei, wenn sie nicht mehr verwendet werden, um zu verhindern, dass die UltraLite-Datenbankdatei in Verwendung bleibt, solange eine Verbindung zur Datenbank besteht.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Rufen Sie die Close-Methode für die Freigabe der Ressourcen auf.

   Verwenden Sie den folgenden Code, wenn die Anwendung eine Verbindung zu der Datenbank nicht mehr benötigt:


```
if( conn != NULL ) {
    conn->Close( &ulerr );
}
```

2. Rufen Sie die Fini-Methode zum Finalisieren des ULDatabaseManager-Objekts auf.

   Verwenden Sie den folgenden Code beim Schließen der Anwendung.

   ```
   ULDatabaseManager.Fini();
   ```

Ergebnisse

Die Datenbankverbindung wird geschlossen und die Ressourcen werden freigegeben.

Siehe auch

- ULConnection.Close-Methode [UltraLite C++] auf Seite 112
- ULDatabaseManager.Fini-Methode [UltraLite C++] auf Seite 142

So erstellen und Sie UltraLite C++-Anwendungen und führen ein Deployment durch.

Beim Erstellen einer C/C++-Anwendung, die nicht die UltraLite-Engine verwendet, können Sie entweder eine Verknüpfung zu einer statischen UltraLite-Laufzeitbibliothek herstellen (durch diese Methode wird sichergestellt, dass der gesamte UltraLite-Code mit Ihrer Anwendung verknüpft wird) oder Sie können (unter Windows und Windows Mobile) eine Verknüpfung zu einer Importbibliothek herstellen und den UltraLite-Laufzeitcode dynamisch laden, wenn die Anwendung startet.

Linker/Compiler-Optionen zum Erstellen und Verknüpfen von Laufzeitumgebungen für das Linux-Deployment

Die linker/compiler-Optionen für *libulrt.a* sind:

```
-L<$SQLANY16>/ultralite/linux/x86/586/lib -lulrt -|ulbase
```

Optionen für die Engine lauten:

```
-L<$SQLANY16>/ultralite/linux/x86/586/lib -lulrtc -|ulbase
```

Der Header-Befehlszeilenoption lautet:

```
-I<$SQLANY16>/sdk/include
```

Deployment einer UltraLite-Anwendung für Windows Mobile (statische Verknüpfung)

Geben Sie die entsprechenden Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter, Protokolloptionen, Verknüpfungsbibliotheken, Methodenaufrufe und Deployment-Dateien an, um sicherzustellen, dass Ihre UltraLite C++-Anwendung auf Windows- und Windows Mobile-Geräten ausgeführt werden kann.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Geben Sie die folgenden Parameter an:
   - Wenn Sie Verschleierung verwenden, setzen Sie beim Erstellen der Datenbank den **obfuscate=1**-Erstellungsparameter.
   - Wenn Sie AES- oder FIPS 140-2 AES-Verschlüsselung verwenden, setzen Sie beim Erstellen der Datenbank oder beim Herstellen einer Verbindung mit der Datenbank den Verbindungsparameter **DBKEY=***encryption-key*.
2. Legen Sie die entsprechenden Parametereinstellungen fest, wenn Sie die Synchronisation in Ihrer UltraLite-Anwendung verwenden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Setzen Sie den Synchronisationsparameter **Stream** auf **tcpip**.
HTTP	Setzen Sie den Synchronisationsparameter **Stream** auf **http**.
RSA_TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**.
RSA HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**.
RSA FIPS 140-2 TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**. Setzen Sie die Protokolloption **fips=yes**.
RSA FIPS 140-2 HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**. Setzen Sie die Protokolloption **fips=yes**.

3. Wenn Sie RSA- oder FIPS 140-2 RSA-Ende-zu-Ende-Verschlüsselung verwenden, setzen Sie die Protokolloption **e2ee_public_key=***key-file*.
4. Wenn Sie ZLIB-Kompression verwenden, setzen Sie die Protokolloption **compression=zlib**.
5. Erstellen Sie Verknüpfungen mit den folgenden Dateien:
   - *ulrt.lib*.
   - *ulbase.lib*.
   - Bei Verwendung von RSA TLS- oder RSA HTTPS-Synchronisation *ulrsa.lib*.

   Für Windows Mobile befinden sich diese Dateien in *%SQLANY16%\UltraLite\CE\Arm.50\Lib*. Für Windows befinden sie sich unter *%SQLANY16%\UltraLite\Windows\x64\Lib\VS9* oder *%SQLANY16%\UltraLite\Windows\x86\Lib\VS9*.

6. Rufen Sie in Ihrer UltraLite-Anwendung die folgenden Methoden auf:
   - Bei Verwendung von AES-Verschlüsselung die ULDatabaseManager.EnableAesDBEncryption-Methode.
   - Bei Verwendung von FIPS 140-2 AES-Verschlüsselung die ULDatabaseManager.EnableAesFipsDBEncryption-Methode.
7. Stellen Sie sicher, dass die folgenden Methoden für den in Ihrer UltraLite-Anwendung verwendeten Synchronisationstyp aufgerufen werden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Rufen Sie die EnableTcpipSynchronization-Methode auf.
HTTP	Rufen Sie die EnableHttpSynchronization-Methode auf.
RSA_TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaSyncEncryption auf.
RSA HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaSyncEncryption auf.
RSA FIPS 140-2 TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaFipsEncryption auf.
RSA FIPS 140-2 HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaFipsSyncEncryption auf.

8. Stellen Sie die folgenden Dateien bereit:
   - Bei Verwendung von FIPS 140-2 AES-Verschlüsselung *ulfips16.dll* und *sbgse2.dll*.
   - Bei Verwendung von RSA FIPS 140-2 TLS- oder RSA FIPS 140-2 HTTPS-Synchronisation *sbgse2.dll* und *mlcrsafips16.dll*.

   Für Windows Mobile befinden sich die Dateien in *%SQLANY16%\UltraLite\CE\Arm.50*. Für Windows befinden sich die Dateien in *%SQLANY16%\UltraLite\Windows\x64* oder *%SQLANY16%\UltraLite\Windows\x86*.

Ergebnisse

Die UltraLite-C++-Anwendung mit statischer Verknüpfung wird auf Windows-Desktop-Computern oder Windows Mobile-Geräten, auf denen sie bereitgestellt wurde, erfolgreich ausgeführt.

Nächste Schritte

Führen Sie das Deployment einer UltraLite-Datenbank auf den Windows-Desktop-Computer oder das Windows Mobile-Gerät durch, auf denen die Anwendung bereitgestellt wurde, oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [*UltraLite - Datenbankverwaltung*]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

Deployment einer UltraLite-Anwendung für Windows Mobile (dynamische Verknüpfung)

Geben Sie die entsprechenden Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter, Protokolloptionen, Verknüpfungsbibliotheken, Methodenaufrufe und Deployment-Dateien an, um sicherzustellen, dass Ihre UltraLite C++-Anwendung auf Windows- und Windows Mobile-Geräten ausgeführt werden kann.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Geben Sie die folgenden Parameter an:

   - Wenn Sie Verschleierung verwenden, setzen Sie beim Erstellen der Datenbank den **obfuscate=1**-Erstellungsparameter.
   - Wenn Sie AES- oder FIPS 140-2 AES-Verschlüsselung verwenden, setzen Sie beim Erstellen der Datenbank oder beim Herstellen einer Verbindung mit der Datenbank den Verbindungsparameter **DBKEY=***encryption-key*.

2. Legen Sie die entsprechenden Parametereinstellungen fest, wenn Sie die Synchronisation in Ihrer UltraLite-Anwendung verwenden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Setzen Sie den Synchronisationsparameter **Stream** auf **tcpip**.
HTTP	Setzen Sie den Synchronisationsparameter **Stream** auf **http**.
RSA_TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**.
RSA HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**.
RSA FIPS 140-2 TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**. Setzen Sie die Protokolloption **fips=yes**.

Synchronisationstyp	Parametereinstellungen
RSA FIPS 140-2 HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**. Setzen Sie die Protokolloption **fips=yes**.

3. Wenn Sie RSA- oder FIPS 140-2 RSA-Ende-zu-Ende-Verschlüsselung verwenden, setzen Sie die Protokolloption **e2ee_public_key**=*key-file*.

4. Wenn Sie ZLIB-Kompression verwenden, setzen Sie die Protokolloption **compression=zlib**.

5. Erstellen Sie Verknüpfungen mit den folgenden Dateien:

   - *ulbase.lib*
   - *ulimp.lib*

   Für Windows Mobile befinden sich diese Dateien in *%SQLANY16%\UltraLite\CE\Arm.50\Lib*. Für Windows befinden sie sich unter *%SQLANY16%\UltraLite\Windows\x64\Lib\VS9* oder *%SQLANY16%\UltraLite\Windows\x86\Lib\VS9*.

6. Wenn eine Verknüpfung mit der *ulimp.lib*-Bibliothek erstellt wird, definieren Sie den UL_USE_DLL-Präprozessormakro beim Kompilieren. Beispielsweise kann die Eingabe so lauten:

   ```
   -DUL_USE_DLL
   ```

7. Rufen Sie in Ihrer UltraLite-Anwendung die folgenden Methoden auf:

   - Bei Verwendung von AES-Verschlüsselung die ULDatabaseManager.EnableAesDBEncryption-Methode.
   - Bei Verwendung von FIPS 140-2 AES-Verschlüsselung die ULDatabaseManager.EnableAesFipsDBEncryption-Methode.

8. Stellen Sie sicher, dass die folgenden Methoden für den in Ihrer UltraLite-Anwendung verwendeten Synchronisationstyp aufgerufen werden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Rufen Sie die EnableTcpipSynchronization-Methode auf.
HTTP	Rufen Sie die EnableHttpSynchronization-Methode auf.
RSA_TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaSyn-cEncryption auf.
RSA HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsa-SyncEncryption auf.

Synchronisationstyp	Parametereinstellungen
RSA FIPS 140-2 TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaFipsEncryption auf.
RSA FIPS 140-2 HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaFipsSyncEncryption auf.

9. Stellen Sie die folgenden Dateien bereit:

   - *ulrt16.dll*.
   - Wenn Sie ZLIB-Kompression verwenden, *mlczlib16.dll*.
   - Bei der Verwendung von RSA TLS, RSA HTTPS oder RSA E2EE *mlcrsa16.dll*.
   - Bei Verwendung von FIPS 140-2 AES-Verschlüsselung *ulfips16.dll* und *sbgse2.dll*.
   - Bei Verwendung von RSA FIPS 140-2 TLS- oder RSA FIPS 140-2 HTTPS-Synchronisation *sbgse2.dll* und *mlcrsafips16.dll*.

   Für Windows Mobile befinden sich die Dateien in *%SQLANY16%\UltraLite\CE\Arm.50*. Für Windows befinden sich die Dateien in *%SQLANY16%\UltraLite\Windows\x64* oder *%SQLANY16%\UltraLite\Windows\x86*.

Ergebnisse

Die UltraLite-C++-Anwendung mit dynamischer Verknüpfung wird auf Windows-Desktop-Computern oder Windows Mobile-Geräten, auf denen sie bereitgestellt wurde, erfolgreich ausgeführt.

Nächste Schritte

Führen Sie das Deployment einer UltraLite-Datenbank auf den Windows-Desktop-Computer oder das Windows Mobile-Gerät durch, auf denen die Anwendung bereitgestellt wurde, oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [*UltraLite - Datenbankverwaltung*]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

Deployment einer UltraLite-Anwendung für Windows Mobile (UltraLite-Engine)

Geben Sie die entsprechenden Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter, Protokolloptionen, Verknüpfungsbibliotheken, Methodenaufrufe und Deployment-Dateien an, um sicherzustellen, dass Ihre UltraLite C++-Anwendung auf Windows- und Windows Mobile-Geräten ausgeführt werden kann.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Geben Sie die folgenden Parameter an:
   - Wenn Sie Verschleierung verwenden, setzen Sie beim Erstellen der Datenbank den **obfuscate=1**-Erstellungsparameter.
   - Wenn Sie AES- oder FIPS 140-2 AES-Verschlüsselung verwenden, setzen Sie beim Erstellen der Datenbank oder beim Herstellen einer Verbindung mit der Datenbank den Verbindungsparameter **DBKEY=***encryption-key*.
2. Legen Sie die entsprechenden Parametereinstellungen fest, wenn Sie die Synchronisation in Ihrer UltraLite-Anwendung verwenden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Setzen Sie den Synchronisationsparameter **Stream** auf **tcpip**.
HTTP	Setzen Sie den Synchronisationsparameter **Stream** auf **http**.
RSA_TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**.
RSA HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**.
RSA FIPS 140-2 RSA TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**. Setzen Sie die Protokolloption **fips=yes**.
RSA FIPS 140-2 RSA HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**. Setzen Sie die Protokolloption **fips=yes**.

3. Wenn Sie RSA- oder FIPS 140-2 RSA-Ende-zu-Ende-Verschlüsselung verwenden, setzen Sie die Protokolloption **e2ee_public_key=***key-file*.
4. Wenn Sie ZLIB-Kompression verwenden, setzen Sie die Protokolloption **compression=zlib**.
5. Erstellen Sie Verknüpfungen mit den folgenden Dateien:
   - *ulrtc.lib*
   - *ulbase.lib*

   Für Windows Mobile befinden sich diese Dateien in *%SQLANY16%\UltraLite\CE\Arm.50\Lib*. Für Windows befinden sie sich unter *%SQLANY16%\UltraLite\Windows\x64\Lib\VS9* oder *%SQLANY16%\UltraLite\Windows\x86\Lib\VS9*.

6. Stellen Sie sicher, dass die folgenden Methoden für den in Ihrer UltraLite-Anwendung verwendeten Synchronisationstyp aufgerufen werden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Rufen Sie die EnableTcpipSynchronization-Methode auf.
HTTP	Rufen Sie die EnableHttpSynchronization-Methode auf.
RSA_TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaSyncEncryption auf.
RSA HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaSyncEncryption auf.
RSA FIPS 140-2 TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaFipsEncryption auf.
RSA FIPS 140-2 HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaFipsSyncEncryption auf.

7. Stellen Sie die folgenden Dateien bereit:

   - *uleng16.exe*.
   - Wenn Sie ZLIB-Kompression verwenden, *mlczlib16.dll*.
   - Bei Verwendung von RSA TLS, RSA HTTPS oder RSA E2EE *mlcrsa16.dll*.
   - Bei Verwendung von FIPS 140-2 AES-Verschlüsselung *ulfips16.dll* und *sbgse2.dll*.
   - Bei Verwendung von RSA FIPS 140-2 TLS, RSA FIPS 140-2 HTTPS oder RSA FIPS 140-2 E2EE *sbgse2.dll* und *mlcrsafips16.dll*.

   Für Windows Mobile befinden sich die Dateien in *%SQLANY16%\UltraLite\CE\Arm.50*. Für Windows befinden sich die Dateien in *%SQLANY16%\UltraLite\Windows\x64* oder *%SQLANY16%\UltraLite\Windows\x86*.

Ergebnisse

Die UltraLite-C++-Anwendung, die die UltraLite Engine verwendet, wird auf Windows-Desktop-Computern oder Windows Mobile-Geräten, auf denen sie bereitgestellt wurde, erfolgreich ausgeführt.

Nächste Schritte

Führen Sie das Deployment einer UltraLite-Datenbank auf den Windows-Desktop-Computer oder das Windows Mobile-Gerät durch, auf denen die Anwendung bereitgestellt wurde, oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [*UltraLite - Datenbankverwaltung*]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

Deployment einer UltraLite-Anwendung für Mac OS X oder iOS

Geben Sie die entsprechenden Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter, Protokolloptionen, Methodenaufrufe und Deployment-Dateien an, um sicherzustellen, dass die UltraLiteJ-Anwendung auf Mac-Computern, iPhones oder iPads erfolgreich ausgeführt wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Fügen Sie für Mac OS X folgende Laufzeitbibliothekdateien zu Ihrem Xcode-Projekt hinzu:

   - */Applications/SQLAnywhere16/System/ultralite/macosx/x86_64/libulrt.a*
   - */Applications/SQLAnywhere16/System/ultralite/macosx/x86_64/libulbase.a*

2. Für iOS führen Sie die Verknüpfung mit der UltraLite-Laufzeitbibliothek folgendermaßen durch:

   Fügen Sie *install-dir/ultralite/iphone/libulrt.a* zur **Frameworks**-Gruppe in Xcode hinzu.

   ODER

   Fügen Sie Folgendes in die Build-Einstellung **Other Linker Flags** (*OTHER_LDFLAGS*) ein:

   ```
   -L$(SQLANY_ROOT)/ultralite/iphone
   -lulrt
   ```

   Dabei gilt: *SQLANY_ROOT* ist eine benutzerdefinierte Build-Einstellung, die auf das Installationsverzeichnis von SQL Anywhere verweist.

   UltraLite-Laufzeitdateien müssen nach der Installation erstellt werden. Folgen Sie den Anweisungen in *install-dir/ultralite/iphone/readme.txt*.

3. Fügen Sie die geeigneten Frameworks zu Ihrem Xcode-Projekt hinzu:

   - Für Mac OS X *Frameworks CoreFoundation.framework*, *CoreServices.framework* und *Security.framework*.
   - Für iOS, *CFNetwork.framework* und *Security.framework*.

4. Geben Sie die folgenden Parameter an:

   - Wenn Sie Verschleierung verwenden, setzen Sie beim Erstellen der Datenbank den **obfuscate=1**-Erstellungsparameter.

- Wenn Sie AES-Verschlüsselung verwenden, setzen Sie beim Erstellen der Datenbank oder beim Herstellen einer Verbindung mit der Datenbank den Verbindungsparameter **DBKEY=***Chiffrierschlüssel*.

5. Legen Sie die entsprechenden Parametereinstellungen fest, wenn Sie die Synchronisation in Ihrer UltraLite-Anwendung verwenden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Setzen Sie den Synchronisationsparameter **Stream** auf **tcpip**.
HTTP	Setzen Sie den Synchronisationsparameter **Stream** auf **http**.
RSA_TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**.
RSA HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**.

6. Wenn Sie RSA E2EE-Verschlüsselung verwenden, setzen Sie die Protokolloption **e2ee_public_key=***key-file*

7. Wenn Sie ZLIB-Kompression verwenden, setzen Sie die Protokolloption **compression=zlib**.

8. Wenn Sie AES-Verschlüsselung verwenden, rufen Sie die ULDatabaseManager.EnableAesDBEncryption-Methode auf.

9. Stellen Sie sicher, dass die folgenden Methoden für den in Ihrer UltraLite-Anwendung verwendeten Synchronisationstyp aufgerufen werden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Rufen Sie die EnableTcpipSynchronization-Methode auf.
HTTP	Rufen Sie die EnableHttpSynchronization-Methode auf.
RSA_TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaSyncEncryption auf.
RSA HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaSyncEncryption auf.

Ergebnisse

Die UltraLite-Anwendung wird auf Mac OS X-Desktop-Computern oder iOS-Geräten, auf denen sie bereitgestellt wurde, erfolgreich ausgeführt.

Nächste Schritte

Führen Sie das Deployment einer UltraLite-Datenbank auf Mac-Desktop-Computer, iPhone oder iPad durch, auf denen die Anwendung bereitgestellt wurde, oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [*UltraLite - Datenbankverwaltung*]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

Deployment einer UltraLite-Anwendung für Linux

Geben Sie die entsprechenden Erstellungsparameter, Verbindungsparameter, Synchronisationsparameter, Protokolloptionen, Methodenaufrufe und Deployment-Dateien an, um sicherzustellen, dass die UltraLiteJ-Anwendung unter Linux erfolgreich ausgeführt wird.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Geben Sie die folgenden Parameter an:
   - Wenn Sie Verschleierung verwenden, setzen Sie beim Erstellen der Datenbank den **obfuscate=1**-Erstellungsparameter.
   - Wenn Sie AES-Verschlüsselung verwenden, setzen Sie beim Erstellen der Datenbank oder beim Herstellen einer Verbindung mit der Datenbank den Verbindungsparameter **DBKEY=***encryption-key*.
2. Legen Sie die entsprechenden Parametereinstellungen fest, wenn Sie die Synchronisation in Ihrer UltraLite-Anwendung verwenden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Setzen Sie den Synchronisationsparameter **Stream** auf **tcpip**.
HTTP	Setzen Sie den Synchronisationsparameter **Stream** auf **http**.
RSA_TLS	Setzen Sie den Synchronisationsparameter **Stream** auf **tls**.
RSA HTTPS	Setzen Sie den Synchronisationsparameter **Stream** auf **https**.

3. Wenn Sie RSA- oder FIPS 140-2 RSA-Ende-zu-Ende-Verschlüsselung verwenden, setzen Sie die Protokolloption **e2ee_public_key=***key-file*.
4. Wenn Sie ZLIB-Kompression verwenden, setzen Sie die Protokolloption **compression=zlib**.
5. Erstellen Sie Verknüpfungen mit den folgenden Dateien:
   - *libulrt.a*.

- *libulbase.a*.
- Bei Verwendung von RSA TLS, RSA HTTPS oder RSA E2EE *libulrsa.a*.

Diese Dateien befinden sich in */opt/sqlanywhere16/ultralite/linux/x86/586/lib*.

6. Wenn Sie AES-Verschlüsselung verwenden, rufen Sie die ULDatabaseManager.EnableAesDBEncryption-Methode auf.

7. Stellen Sie sicher, dass die folgenden Methoden für den in Ihrer UltraLite-Anwendung verwendeten Synchronisationstyp aufgerufen werden:

Synchronisationstyp	Parametereinstellungen
TCP/IP	Rufen Sie die EnableTcpipSynchronization-Methode auf.
HTTP	Rufen Sie die EnableHttpSynchronization-Methode auf.
RSA_TLS	Rufen Sie die Methoden EnableTlsSynchronization und EnableRsaSyn-cEncryption auf.
RSA HTTPS	Rufen Sie die Methoden EnableHttpsSynchronization und EnableRsaSyn-cEncryption auf.

Ergebnisse

Die UltraLite-Anwendung wird erfolgreich auf dem Linux-Computer ausgeführt, auf dem Sie das Deployment durchgeführt haben.

Nächste Schritte

Führen Sie das Deployment einer UltraLite-Datenbank auf den Linux-Computer durch, auf dem die Anwendung bereitgestellt wurde, oder erstellen Sie eine neue Datenbank mit der per Deployment bereitgestellten Anwendung.

Siehe auch

- „Kompilierungs- und Deploymentspezifikationen für UltraLite-Anwendungen“ [*UltraLite - Datenbankverwaltung*]
- „Datenbank-Deploymenttechniken für UltraLite und UltraLite Java Edition“ [*UltraLite - Datenbankverwaltung*]

UltraLite C++-Anwendungsentwicklung mit Embedded SQL

Dieser Abschnitt beschreibt die Erstellung des Datenbankzugriffscode für Embedded SQL UltraLite-Anwendungen.

Siehe auch

- „UltraLite Embedded SQL API-Referenz“ auf Seite 227
- „UltraLite-Dienstprogramm für den SQL-Präprozessor (sqlpp)“ [*UltraLite - Datenbankverwaltung*]

Kurzanleitung für die Embedded SQL-Anwendungsentwicklung für UltraLite

Wenn Sie Embedded SQL-Anwendungen entwickeln, mischen Sie SQL-Anweisungen mit Standard C- oder C++-Quellcode. Um Embedded SQL-Anwendungen zu entwickeln, müssen Sie die Programmiersprachen C oder C++ kennen.

Der Entwicklungsprozess für Embedded SQL-Anwendungen sieht wie folgt aus:

1. Entwerfen Sie Ihre UltraLite-Datenbank.
2. Schreiben Sie Ihren Quellcode in eine Embedded SQL-Quellcodedatei, die in der Regel die Erweiterung *.sqc* hat.

   Wenn Sie Datenzugriff in Ihrem Quellcode benötigen, benutzen Sie die SQL-Anweisung, die Sie ausführen wollen, mit den EXEC SQL-Schlüsselwörtern als Präfix. Beispiel:

   ```
   EXEC SQL BEGIN DECLARE SECTION
      int cost
      char pname[31];
   EXEC SQL END DECLARE SECTION

   EXEC SQL SELECT price, prod_name
      INTO :cost, :pname
      FROM ULProduct
      WHERE prod_id= :pid;
   ```
3. Verarbeiten Sie die *.sqc*-Dateien im Präprozessor.

   SQL Anywhere enthält einen SQL-Präprozessor (sqlpp), der die *.sqc*-Dateien liest und *.cpp*-Dateien generiert. Diese Dateien enthalten Funktionsaufrufe an die UltraLite-Laufzeitbibliothek.
4. Kompilieren Sie Ihre *.cpp*-Dateien.
5. Verknüpfen Sie die *.cpp*-Dateien.

   Sie müssen die Dateien mit der UltraLite-Laufzeitbibliothek verknüpfen.

Siehe auch

- „Erstellen von Embedded SQL-Anwendungen“ auf Seite 65
- „UltraLite C++-Anwendungsentwicklung mit Embedded SQL“ auf Seite 35

Embedded SQL-Beispiel

Embedded SQL ist eine Umgebung, die eine Kombination aus C/C++-Programmcode und Pseudocode darstellt. Der Pseudocode, der mit traditionellem C/C++-Code durchsetzt sein kann, ist eine Teilmenge

von SQL-Anweisungen. Ein Präprozessor konvertiert die Embedded SQL-Anweisungen in Funktionsaufrufe, die Teil des tatsächlichen Codes sind, der zur Erstellung der Anwendung kompiliert wird.

Es folgt ein sehr einfaches Beispiel für ein Embedded SQL-Programm. Es veranschaulicht die Aktualisierung einer UltraLite-Datenbank, indem der Nachname von Mitarbeiter 195 geändert wird.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main( )
{
   db_init( &sqlca );
   EXEC SQL WHENEVER SQLERROR GOTO error;
   EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
   EXEC SQL UPDATE employee
      SET emp_lname = 'Johnson'
      WHERE emp_id = 195;
   EXEC SQL COMMIT;
   EXEC SQL DISCONNECT;
   db_fini( &sqlca );
   return( 0 );
   error:
      printf( "update unsuccessful: sqlcode = %ld\n",
         sqlca.sqlcode );
      return( -1 );
}
```

Dieses Beispiel ist zwar zu einfach, um tatsächlich verwendet werden zu können, doch es veranschaulicht folgende Aspekte, die für alle Embedded SQL-Anwendungen gleichermaßen gelten:

- Jede SQL-Anweisung beginnt mit den Schlüsselwörtern EXEC SQL.
- Jede SQL-Anweisung endet mit einem Strichpunkt.
- Einige Embedded SQL-Anweisungen sind in Standard-SQL nicht enthalten. Ein Beispiel ist die Anweisung INCLUDE SQLCA.
- Zusätzlich zu SQL-Anweisungen stellt Embedded SQL auch Bibliotheksfunktionen für die Ausführung einiger spezifischer Aufgaben bereit. Die Funktionen db_init und db_fini sind zwei Beispiele von Aufrufen von Bibliotheksfunktionen.

Initialisierung

Der obige Beispielcode veranschaulicht Initialisierungsanweisungen, die einbezogen werden müssen, bevor die Daten in der UltraLite-Datenbank bearbeitet werden:

1. Legen Sie den SQL-Kommunikationsbereich (SQLCA) mit folgendem Befehl fest:

   ```
   EXEC SQL INCLUDE SQLCA;
   ```

   Diese Definition muss die erste Embedded SQL-Anweisung sein. Der logische Platz für den Befehl ist das Ende der Include-Liste.

   Wenn Ihre Anwendung über mehrere *.sqc*-Dateien verfügt, muss diese Zeile in jeder der Dateien enthalten sein.

2. Die erste Aktion im Datenbankprogramm muss der Aufruf der Embedded SQL-Bibliotheksfunktion db_init sein. Diese Funktion initialisiert die UltraLite-Laufzeitbibliothek. Vor diesem Aufruf dürfen nur Embedded SQL-Definitionsanweisungen ausgeführt werden.

3. Sie müssen mit der Anweisung SQL CONNECT eine Verbindung zur UltraLite-Datenbank herstellen.

Programmende vorbereiten

Der oben genannte Beispielcode veranschaulicht die Sequenz von Aufrufen, die erforderlich sind, um das Programm zu beenden:

1. Schreiben Sie noch ausstehende Änderungen fest oder setzen Sie sie zurück.

2. Trennen Sie die Verbindung mit der Datenbank.

3. Beenden Sie Ihre SQL-Arbeit mit einem Aufruf der Bibliotheksmethode db_fini.

Wenn Sie das Programm beenden, werden alle noch nicht festgeschriebenen Datenbankänderungen automatisch zurückgesetzt.

Fehlerbehandlung

Es gibt praktisch keine Interaktion zwischen dem SQL- und dem C-Code in diesem Beispiel. Der C-Code steuert nur den Datenfluss des Programms. Die Anweisung WHENEVER wird zur Fehlerprüfung verwendet. Die Fehleraktion, in diesem Beispiel GOTO, wird immer ausgeführt, wenn eine SQL-Anweisung einen Fehler verursacht.

Siehe auch

- „db_init-Methode“ auf Seite 228

Embedded SQL-Programmstruktur

Alle Embedded SQL-Anweisungen beginnen mit den Worten EXEC SQL und enden mit einem Semikolon. Normale Kommentare der Sprache C sind innerhalb von Embedded SQL-Anweisungen erlaubt.

Der Quellcode eines C-Programms, das Embedded SQL benutzt, muss vor der ersten Embedded SQL-Anweisung in der Quelldatei folgende Anweisung enthalten:

```
EXEC SQL INCLUDE SQLCA;
```

Die erste Embedded SQL-Programmanweisung, die in einem Programm ausgeführt wird, muss eine SQL CONNECT-Anweisung sein. Die CONNECT-Anweisung liefert Verbindungsparameter, die verwendet werden, um eine Verbindung mit der UltraLite-Datenbank herzustellen.

Einige Embedded SQL-Befehle erzeugen keinen C-Programmcode oder benötigen keine Verbindung zur Datenbank. Nur folgende Befehle sind vor der CONNECT-Anweisung erlaubt. Die wichtigsten sind die INCLUDE-Anweisung und die WHENEVER-Anweisung für die Fehlerbehandlung.

SQL Communications Area initialisieren

Der SQL-Kommunikationsbereich (SQLCA) ist ein Speicherbereich, der für die Kommunikation von Statistiken und Fehlern von der Anwendung zur Datenbank und zurück an die Anwendung verwendet wird. Der SQLCA-Bereich wird als Handle für die Kommunikationsverbindung zwischen Anwendung und Datenbank benutzt. Er wird explizit an alle Bibliotheksfunktionen der Datenbank übergeben, die mit der Datenbank kommunizieren. Implizit wird er an alle Embedded SQL-Anweisungen übergeben.

UltraLite definiert im generierten Code eine globale SQLCA-Variable. Der Präprozessor erzeugt eine externe Referenz für die globale SQLCA-Variable. Die externe Referenz heißt sqlca und ist vom Typ SQLCA. Die globale Variable selbst wird in der Importbibliothek deklariert.

Der SQLCA-Typ wird in der Header-Datei *%SQLANY16%\SDK\Include\sqlca.h* definiert.

Nach der Deklaration des SQLCA-Bereichs (`EXEC SQL INCLUDE SQLCA;`), aber bevor Ihre Anwendung Vorgänge in der Datenbank durchführen kann, müssen Sie den Kommunikationsbereich durch den Aufruf von db_init und die Übergabe des SQLCA-Bereichs initialisieren:

```
db_init( &sqlca );
```

SQLCA-Bereich bietet Fehlercodes

Sie referenzieren den SQLCA-Bereich, um einen bestimmten Fehlercode zu testen. Das Feld sqlcode enthält einen Fehlercode, wenn eine Datenbankanforderung einen Fehler verursacht. Makros werden zur Referenzierung des Felds sqlcode und einiger anderer Felder im SQLCA-Bereich definiert.

SQLCA-Felder

Der SQLCA-Bereich enthält die folgenden Felder:

- **sqlcaid** Ein 8-Byte-Zeichenfeld, das die Zeichenfolge SQLCA zur Identifizierung der SQLCA-Struktur enthält. Dieses Feld unterstützt die Fehlersuche, wenn Sie Speicherinhalte untersuchen.
- **sqlcabc** Ganzzahl, die die Länge der SQLCA-Struktur in Byte enthält
- **sqlcode** Eine Ganzzahl, die einen Fehlercode enthält, wenn die Datenbank einen Fehler bei einer Anforderung feststellt. Definitionen für den Fehlercode sind in der Header-Datei *%SQLANY16%\SDK\Include\sqlerr.h* enthalten. Der Fehlercode für einen erfolgreichen Vorgang ist 0 (null), für eine Warnung ist der Fehlercode positiv und für einen Fehler ist er negativ.

  Sie können auf dieses Feld direkt mit dem Makro SQLCODE zugreifen.
- **sqlerrml** Die Länge der Daten im Feld sqlerrmc.

  UltraLite-Anwendungen verwenden dieses Feld nicht.
- **sqlerrmc** Kann einen oder mehrere Zeichenfolgen enthalten, die in eine Fehlermeldung einzufügen sind. Einige Fehlermeldungen enthalten als Platzhalter die Zeichenfolge (*%1*), die durch den Text in diesem Feld ersetzt wird.

UltraLite-Anwendungen verwenden dieses Feld nicht.

- **sqlerrp** Reserviert
- **sqlerrd** Array von Ganzzahlen.
- **sqlwarn** Reserviert

  UltraLite-Anwendungen verwenden dieses Feld nicht.
- **sqlstate** Der Statuswert SQLSTATE.

  UltraLite-Anwendungen verwenden dieses Feld nicht.

Siehe auch

- „SQL Anywhere - Fehlermeldungen“ [*Fehlermeldungen*]

UltraLite-Datenbankverbindung mithilfe von Embedded SQL verwenden

Um eine Verbindung mit einer UltraLite-Datenbank aus einer Embedded SQL-Anwendung einzurichten, fügen Sie die EXEC SQL CONNECT-Anweisung nach dem Initialisieren des SQLCA in Ihren Code ein.

Die CONNECT-Anweisung hat folgende Form:

EXEC SQL CONNECT USING
'uid=*user-name***;pwd=***password***;dbf=***database-filename***'**;

Die Verbindungszeichenfolge (zwischen Apostrophen) kann zusätzliche Datenbank-Verbindungsparameter enthalten.

Wenn Sie in Ihrer Anwendung mehr als eine Datenbankverbindung benötigen, können Sie mehrere SQLCAs oder einen einzelnen SQLCA für die Verwaltung der Verbindungen verwenden.

Verwenden eines einzelnen SQLCA-Bereichs

Sie können einen einzelnen SQLCA-Bereich für die Verwaltung mehrerer Verbindungen zu einer Datenbank verwenden.

Jeder SQLCA hat eine einzelne aktive oder laufende Verbindung, aber diese Verbindung kann geändert werden. Bevor Sie einen Befehl ausführen, benutzen Sie die SET CONNECTION-Anweisung, um die Verbindung anzugeben, auf der der Befehl ausgeführt werden soll.

Siehe auch

- „UltraLite-Verbindungsparameter“ [*UltraLite - Datenbankverwaltung*]
- „CONNECT-Anweisung [ESQL] [Interactive SQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Mehrere SQLCA-Bereiche verwenden, um mehrere Datenbankverbindungen zu verwalten

Verwenden Sie die Embedded SQL-Anweisung SET SQLCA, um den SQL-Präprozessor anzuweisen, einen spezifischen SQLCA-Bereich für Datenbankanforderungen zu verwenden.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Initialisieren Sie jeden SQLCA-Bereich, der in Ihrem Programm benutzt wird, mit einem Aufruf von **db_init**.
2. Setzen Sie die SQLCA-Referenz zu Beginn eines Programms oder in einer Header-Datei auf Task-spezifische Daten:

   ```
   EXEC SQL SET SQLCA 'task_data->sqlca';
   ```

   Diese Anweisung generiert keinen Code und wirkt sich nicht auf die Performance aus.

Ergebnisse

Der Status innerhalb des Präprozessors wird geändert, sodass jede Referenz auf den SQLCA-Bereich die angegebene Zeichenfolge benutzt.

Nächste Schritte

Bereinigen Sie die Ressourcen beim Schließen Ihrer Anwendung durch einen Aufruf der **db_fini**-Methode.

Siehe auch

- „db_init-Methode“ auf Seite 228
- „SET SQLCA-Anweisung [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „SET CONNECTION-Anweisung [Interactive SQL] [ESQL]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]

Hostvariablen

Embedded SQL-Anweisungen verwenden Hostvariablen, um der Werte an die Datenbank zu übertragen. Hostvariablen sind C-Variablen, die dem SQL-Präprozessor in einem Deklarationsabschnitt bekannt gemacht werden.

Hostvariablendeklaration

Legen Sie die Hostvariablen fest, indem Sie sie in einen Deklarationsabschnitt stellen. Für die Deklaration der Hostvariablen umschließen Sie die normalen Deklarationen der C-Variablen mit den Anweisungen BEGIN DECLARE SECTION und END DECLARE SECTION.

Wenn Sie eine Hostvariable in einer SQL-Anweisung verwenden, müssen Sie dem Variablennamen einen Doppelpunkt (:) als Präfix voranstellen, sodass der Präprozessor erkennt, dass Sie sich auf eine (deklarierte) Hostvariable beziehen und ihn von anderen in der Anweisung zulässigen Bezeichnern unterscheiden kann.

Sie können Hostvariablen anstelle von Konstanten in einer SQL-Anweisung einsetzen. Wenn der Datenbankserver den Befehl ausführt, wird der Wert der Hostvariablen aus den einzelnen Hostvariablen gelesen bzw. in sie geschrieben. Hostvariablen können nicht anstelle von Tabellen- oder Spaltennamen benutzt werden.

Der SQL-Präprozessor durchsucht nur den C-Programmcode, der innerhalb des Deklarationsabschnitts steht. Initialisierer für Variablen sind in einem Deklarationsabschnitt zulässig, während **typedef**-Typen und -Strukturen nicht erlaubt sind.

Das folgende Beispiel zeigt den Gebrauch von Hostvariablen für einen INSERT-Befehl. Variablen werden vom Programm belegt und dann in die Datenbank eingefügt:

```
/* Declare fields for personal data. */
EXEC SQL BEGIN DECLARE SECTION;
   long employee_number = 0;
   char employee_name[50];
   char employee_initials[8];
   char employee_phone[15];
EXEC SQL END DECLARE SECTION;
/* Fill variables with appropriate values. */
/* Insert a row in the database. */
EXEC SQL INSERT INTO Employee
   VALUES (:employee_number, :employee_name,
      :employee_initials, :employee_phone );
```

Datentypen

Um Informationen zwischen einem Programm und dem Datenbankserver austauschen zu können, müssen alle Datenelemente einen Datentyp haben. Eine Hostvariable kann jeden der unterstützten Datentypen haben.

Nur eine begrenzte Anzahl von C-Datentypen werden als Hostvariablen unterstützt. Andererseits haben bestimmte Hostvariablen keinen entsprechenden Datentyp in C.

Mit Makros, die in der Header-Datei *sqlca.h* festgelegt sind, können Sie eine Hostvariable vom Typ VARCHAR, FIXCHAR, BINARY, DECIMAL oder SQLDATETIME deklarieren. Diese Makros werden wie folgt benutzt:

```
EXEC SQL BEGIN DECLARE SECTION;
   DECL_VARCHAR( 10 ) v_varchar;
   DECL_FIXCHAR( 10 ) v_fixchar;
   DECL_BINARY( 4000 ) v_binary;
   DECL_DECIMAL( 10, 2 ) v_packed_decimal;
   DECL_DATETIME v_datetime;
EXEC SQL END DECLARE SECTION;
```

Der Präprozessor erkennt diese Makros innerhalb eines Deklarationsabschnitts und behandelt Variablen ihrem Typ entsprechend.

Folgende Datentypen werden von der Embedded SQL-Programmierschnittstelle unterstützt:

- **16-Bit-Ganzzahl mit Vorzeichen**

```
short int I;
unsigned short int I;
```

- **32-Bit-Ganzzahl mit Vorzeichen**

```
long int l;
unsigned long int l;
```

- **4-Byte-Gleitkommazahl**

```
float f;
```

- **8-Byte-Gleitkommazahl**

```
double d;
```

- **Gepackte Dezimalzahl**

```
DECL_DECIMAL(p,s)
typedef struct TYPE_DECIMAL {
   char array[1];
} TYPE_DECIMAL;
```

- **Zeichenfolge mit Nullabschlusszeichen, mit Leerzeichen aufgefüllt**

```
char a[n]; /* n > 1 */
char *a; /* n = 2049 */
```

  Da das Array der Programmiersprache C auch das Nullabschlusszeichen enthalten muss, wird der Datentyp char a[] dem SQL-Datentyp CHAR(n - 1) zugeordnet, der n-1 Zeichen speichern kann.

  -Hinweis
  Der SQL-Präprozessor geht davon aus, dass ein **Zeiger auf char** auf ein Zeichen-Array von 2049 Byte zeigt und dass dieses Array 2048 Zeichen plus dem Nullabschlusszeichen problemlos speichern kann. Das heißt, der Datentyp "char*" wird auf den SQL-Typ "CHAR(2048)" abgebildet. Ist das nicht der Fall, beschädigt Ihre Anwendung möglicherweise den Speicher.

  Falls Sie einen 16-Bit-Compiler verwenden, können die erforderlichen 2049 Byte einen Überlauf des Programm-Stacks verursachen. Verwenden Sie stattdessen ein deklariertes Array, auch als Parameter einer Funktion, um dem SQL-Präprozessor die Größe des Arrays mitzuteilen. WCHAR und TCHAR verhalten sich ähnlich wie "char".

- **Mit einem Nullabschlusszeichen endende UNICODE- oder breite Zeichenfolge** Jedes Zeichen belegt zwei Byte Platz und kann daher UNICODE-Zeichen enthalten.

```
WCHAR a[n]; /* n > 1 */
```

- **Mit einem Nullabschlusszeichen endende systemabhängige Zeichenfolge** TCHAR entspricht WCHAR für Systeme, die UNICODE als Zeichensatz verwenden (beispielsweise Windows Mobile). Andernfalls entspricht TCHAR dem Datentyp char. Der TCHAR-Datentyp unterstützt Zeichenfolgen in jedem System automatisch.


```
TCHAR a[n]; /* n > 1 */
```

- **Zeichenfolge fester Länge, mit Leerzeichen aufgefüllt**

```
char a; /* n  = 1 */
DECL_FIXCHAR(n) a; /* n >= 1 */
```

- **Zeichenfolge variabler Länge mit 2-Byte-Längenfeld** Wenn Sie dem Datenbankserver Informationen liefern, müssen Sie das Längenfeld setzen. Wenn Sie Informationen vom Datenbankserver abholen, setzt der Datenbankserver das Längenfeld (nicht aufgefüllt).

```
DECL_VARCHAR(n) a; /* n >= 1 */
typedef struct VARCHAR {
   a_sql_ulen len;
   TCHAR array[1];
} VARCHAR;
```

- **Binärdaten variabler Länge mit 2-Byte-Längenfeld** Wenn Sie dem Datenbankserver Informationen liefern, müssen Sie das Längenfeld setzen. Wenn Sie Informationen vom Datenbankserver abrufen, setzt der Datenbankserver das Längenfeld.

```
DECL_BINARY(n) a; /* n >= 1 */
typedef struct BINARY {
  a_sql_ulen len;
   unsigned char array[1];
} BINARY;
```

- **SQLDATETIME-Struktur mit Feldern für jeden Teil eines Zeitstempels**

```
DECL_DATETIME a;
typedef struct SQLDATETIME {
   unsigned short year; /* for example: 1999 */
   unsigned char month; /* 0-11 */
   unsigned char day_of_week; /* 0-6, 0 = Sunday */
   unsigned short day_of_year; /* 0-365 */
   unsigned char day; /* 1-31 */
   unsigned char hour; /* 0-23 */
   unsigned char minute; /* 0-59 */
   unsigned char second; /* 0-59 */
   unsigned long microsecond; /* 0-999999 */
} SQLDATETIME;
```

  Die Struktur SQLDATETIME wird verwendet, um Felder vom Typ DATE, TIME und TIMESTAMP abzurufen (oder einen beliebigen Datentyp, der in einen dieser Datentypen konvertiert werden kann). Anwendungen haben häufig eigene Formate und einen eigenen Programmcode für Zeit und Datum. Das Abrufen von Daten in diese Struktur vereinfacht die Datenbearbeitung für Sie. Die Felder vom Typ DATE, TIME und TIMESTAMP können auch mit einem beliebigen Zeichendatentyp abgerufen und aktualisiert werden.

  Wenn Sie eine SQLDATETIME-Struktur verwenden, um ein Datum, eine Zeit oder einen Zeitstempel in die Datenbank einzugeben, werden die Felder day_of_year und day_of_week ignoriert.

- **DT_LONGVARCHAR** Lange Zeichendaten mit unterschiedlicher Länge. Der Makro definiert wie folgt eine Struktur:

```
#define DECL_LONGVARCHAR( size ) \
  struct { a_sql_uint32    array_len;    \
```

```
  a_sql_uint32    stored_len;   \
  a_sql_uint32    untrunc_len;  \
  char            array[size+1];\
}
```

Die DECL_LONGVARCHAR-Struktur kann mit Daten über 32 kB verwendet werden. Daten können auf einmal oder mithilfe der Anweisung GET DATA in Abschnitten abgerufen werden. Sie können an den Server auf einmal, beziehungsweise in Abschnitten übermittelt werden, indem sie an eine Datenbankvariable mit der SET-Anweisung angehängt werden. Die Daten werden nicht mit einem Nullabschlusszeichen abgeschlossen.

- **DT_LONGBINARY** Lange Binärdaten. Der Makro definiert wie folgt eine Struktur:

```
#define DECL_LONGBINARY( size )  \
  struct { a_sql_uint32    array_len;    \
           a_sql_uint32    stored_len;   \
           a_sql_uint32    untrunc_len;  \
           char            array[size];  \
         }
```

Die DECL_ LONGBINARY-Struktur kann mit Daten von mehr als 32 kB verwendet werden. Daten können auf einmal oder mithilfe der Anweisung GET DATA in Abschnitten abgerufen werden. Sie können an den Server auf einmal, beziehungsweise in Abschnitten übermittelt werden, indem sie an eine Datenbankvariable mit der SET-Anweisung angehängt werden.

Die Strukturen werden in der Datei *%SQLANY16%\SDK\Include\sqlca.h* definiert. Die Typen VARCHAR, BINARY und TYPE_DECIMAL enthalten ein Ein-Zeichen-Array und sind nicht für die Deklaration von Hostvariablen zu gebrauchen. Sie sind allerdings nützlich, um Variablen dynamisch zuzuweisen oder andere Variablen umzuwandeln ("Typecasting").

Datenbank-Datentypen DATE und TIME

Die Embedded SQL-Schnittstelle bietet keine Datentypen, die den verschiedenen Datenbank-Datentypen von DATE und TIME entsprechen. Diese Datenbank-Datentypen werden entweder mit der Struktur SQLDATETIME oder mit Zeichenfolgen abgerufen und aktualisiert.

Es gibt in der Embedded SQL-Schnittstelle keine Datentypen für LONG VARCHAR und LONG BINARY.

Siehe auch

- „Datenbankoptionen“ [*SQL Anywhere Server - Datenbankadministration*]

Hostvariablen verwenden

Hostvariablen können unter folgenden Umständen benutzt werden:

- In einer SELECT-, INSERT-, UPDATE- oder DELETE-Anweisung überall dort, wo eine Zahlen- oder Zeichenfolgenkonstante erlaubt ist.
- In der INTO-Klausel einer SELECT- oder FETCH-Anweisung

- In CONNECT-, DISCONNECT- und SET CONNECT-Anweisungen kann eine Hostvariable anstelle einer Benutzerkennung, eines Kennworts, des Namens einer Verbindung oder einer Datenbank verwendet werden.

Hostvariablen können *nie* anstelle eines Tabellen- oder Spaltennamens benutzt werden.

Hostvariablenbereich

Die Deklaration einer Hostvariablen kann überall dort stehen, wo auch eine C-Variablendeklaration stehen kann. Das gilt auch für den Parameterdeklarationsabschnitt einer C-Funktion. Die dort deklarierten C-Variablen haben ihre normalen Aufgaben (sie stehen innerhalb des Blocks zur Verfügung, in dem sie definiert sind). Da aber der SQL-Präprozessor den C-Code nicht durchsucht, erkennt er keine C-Blöcke.

Präprozessor geht von globalen Hostvariablen aus

Der SQL-Präprozessor geht davon aus, dass Hostvariablen im Anschluss an ihre Deklaration im Quellmodul global bekannt sind. Zwei Hostvariablen können nicht den gleichen Namen haben. Die einzige Ausnahme zu dieser Regel: Zwei Hostvariablen können den gleichen Namen haben, wenn sie vom gleichen Typ sind (einschließlich gegebenenfalls der Länge).

Am Besten erhält jede Hostvariable einen eindeutigen Namen.

Beispiele

Da der SQL-Präprozessor C-Code nicht syntaktisch analysieren kann, geht er davon aus, dass alle Hostvariablen unabhängig davon, wo sie deklariert wurden, im Anschluss an ihre Deklaration global bekannt sind.

```
// Example demonstrating poor coding
EXEC SQL BEGIN DECLARE SECTION;
   long emp_id;
EXEC SQL END DECLARE SECTION;
long getManagerID( void )
{
   EXEC SQL BEGIN DECLARE SECTION;
      long manager_id = 0;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL SELECT manager_id
            INTO :manager_id
            FROM employee
            WHERE emp_number = :emp_id;
   return( manager_number );
}
void setManagerID( long manager_id )
{
   EXEC SQL UPDATE employee
            SET manager_number = :manager_id
            WHERE emp_number = :emp_id;
}
```

Der oben angegebene Code funktioniert zwar, ist aber verwirrend, da der SQL-Präprozessor von der Deklaration in *getManagerID* abhängig ist, wenn die Anweisung in *setManagerID* verarbeitet wird. Sie können diesen Code besser wie folgt schreiben:

```
// Rewritten example
#if 0
```

```
   // Declarations for the SQL preprocessor
   EXEC SQL BEGIN DECLARE SECTION;
      long emp_id;
      long manager_id;
   EXEC SQL END DECLARE SECTION;
#endif
long getManagerID( long emp_id )
{
   long manager_id = 0;
   EXEC SQL SELECT manager_id
            INTO :manager_id
            FROM employee
            WHERE emp_number = :emp_id;
   return( manager_number );
}
void setManagerID( long emp_id, long manager_id )
{
   EXEC SQL UPDATE employee
            SET manager_number = :manager_id
            WHERE emp_number = :emp_id;
}
```

Der SQL-Präprozessor sieht die Deklaration der Hostvariablen innerhalb der #if-Direktive, weil er diese Direktiven ignoriert. Andererseits ignoriert er die Deklarationen innerhalb dieser Prozeduren, da sie sich nicht innerhalb von DECLARE SECTION befinden. Der C-Compiler ignoriert seinerseits die Deklarationen innerhalb der #if-Direktive und verwendet sie innerhalb der Prozeduren.

Diese Deklarationen funktionieren nur, weil Variablen mit demselben Namen deklariert sind und denselben Typ haben.

Ausdrücke als Hostvariablen

Hostvariablen müssen einfache Namen sein, da der SQL-Präprozessor Zeiger oder Referenzausdrücke nicht erkennt. Folgende Anweisung *funktioniert nicht*, weil der SQL-Präprozessor den Punkt-Operator nicht versteht. Dieselbe Syntax hat in SQL eine andere Bedeutung.

```
// Incorrect statement:
EXEC SQL SELECT LAST sales_id INTO :mystruct.mymember;
```

Obwohl die oben dargestellte Syntax nicht zulässig ist, können Sie dennoch mit folgender Methode einen Ausdruck verwenden:

- Binden Sie den SQL-Deklarationsabschnitt in eine #if 0-Präprozessordirektive ein. Der SQL-Präprozessor liest die Deklarationen und verwendet sie für den Rest des Moduls, da er die Präprozessor-Direktiven ignoriert.
- Definieren Sie einen Makro mit dem Namen der Hostvariablen. Da der SQL-Deklarationsabschnitt vom C-Compiler aufgrund der #if-Direktive nicht gesehen wird, kommt es zu keinem Konflikt. Stellen Sie sicher, dass der Makro mit demselben Hostvariabletyp aufgelöst wird.

Der folgende Code veranschaulicht diese Methode, um den Ausdruck *host_value* vor dem SQL-Präprozessor zu verbergen.

```
#include <sqlerr.h>
#include <stdio.h>
```



```
EXEC SQL INCLUDE SQLCA;
typedef struct my_struct {
   long     host_field;
} my_struct;
#if 0
   // Because it ignores #if preprocessing directives,
   // SQLPP reads the following declaration.
   EXEC SQL BEGIN DECLARE SECTION;
      long    host_value;
   EXEC SQL END DECLARE SECTION;
#endif
// Make C/C++ recognize the 'host_value' identifier
// as a macro that expands to a struct field.
#define host_value my_s.host_field
```

Da der SQLPP-Prozessor Direktiven für die bedingte Kompilierung ignoriert, wird *host_value* als Hostvariable vom Datentyp *long* behandelt, und wenn sie nachfolgend als Hostvariable verwendet wird, wird sie diesen Namen ausgeben. Der C/C++-Compiler verarbeitet die ausgegebene Datei und ersetzt alle entsprechenden Verwendungen des Namens durch *my_s.host_field*.

Mit den obigen Deklarationen können Sie auf *host_field* wie folgt zugreifen:

```
void main( void )
{
   my_struct      my_s;
   db_init( &sqlca );
   EXEC SQL CONNECT "DBA" IDENTIFIED BY "SQL";
   EXEC SQL DECLARE my_table_cursor CURSOR FOR
      SELECT int_col FROM my_table order by int_col;
   EXEC SQL OPEN my_table_cursor;
   for( ; ; ) {
      // :host_value references my_s.host_field
      EXEC SQL FETCH NEXT AllRows INTO :host_value;
      if( SQLCODE == SQLE_NOTFOUND ) {
         break;
      }
      printf( "%ld\n", my_s.host_field );
   }
   EXEC SQL CLOSE my_table_cursor;
   EXEC SQL DISCONNECT;
   db_fini( &sqlca );
}
```

Sie können mit derselben Methode andere lvalues-Werte als Hostvariablen verwenden:

- Zeigerumleitungen

  ```
  *ptr
  p_struct->ptr
  (*pp_struct)->ptr
  ```

- Array-Verweise

  ```
  my_array[ I ]
  ```

- Beliebig komplexe lvalues-Werte

Hostvariablen in C++

Eine ähnliche Situation tritt ein, wenn Sie Hostvariablen in C++-Klassen verwenden. Es empfiehlt sich häufig, eine Klasse in einer eigenen Header-Datei zu deklarieren. Diese Header-Datei kann zum Beispiel folgende Deklaration von *my_class* enthalten.

```
typedef short a_bool;
#define  TRUE  ((a_bool)(1==1))
#define  FALSE ((a_bool)(0==1))
public class {
   long  host_member;
   my_class();    // Constructor
   ~my_class();      // Destructor
   a_bool FetchNextRow( void );
      // Fetch the next row into host_member
} my_class;
```

In diesem Beispiel ist jede Methode in einer Embedded SQL-Quelldatei implementiert. Nur einfache Variablen können als Hostvariablen verwendet werden. Die im vorigen Abschnitt vorgestellte Methode kann verwendet werden, um auf ein Datenmitglied einer Klasse zuzugreifen.

```
EXEC SQL INCLUDE SQLCA;
#include "my_class.hpp"
#if 0
   // Because it ignores #if preprocessing directives,
   // SQLPP reads the following declaration.
   EXEC SQL BEGIN DECLARE SECTION;
      long  this_host_member;
   EXEC SQL END DECLARE SECTION;
#endif
// Macro used by the C++ compiler only.
#define this_host_member this->host_member
my_class::my_class()
{
   EXEC SQL DECLARE my_table_cursor CURSOR FOR
      SELECT int_col FROM my_table order by int_col;
   EXEC SQL OPEN my_table_cursor;
}
my_class::~my_class()
{
   EXEC SQL CLOSE my_table_cursor;
}
a_bool my_class::FetchNextRow( void )
{
   // :this_host_member references this->host_member
   EXEC SQL FETCH NEXT AllRows INTO :this_host_member;
   return( SQLCODE != SQLE_NOTFOUND );
}
void main( void )
{
   db_init( &sqlca );
   EXEC SQL CONNECT "DBA" IDENTIFIED BY "SQL";
   {
      my_class mc; // Created after connecting.
      while( mc.FetchNextRow() ) {
         printf( "%ld\n", mc.host_member );
      }
   }
   EXEC SQL DISCONNECT;
   db_fini( &sqlca );
}
```


Im obigen Beispiel wird `this_host_member` für den SQL-Präprozessor deklariert, doch der Makro bewirkt, dass C++ die Variablen in `this->host_member` konvertiert. Der Präprozessor würde sonst den Typ dieser Variablen nicht kennen. Viele C/C++-Compiler tolerieren keine mehrfach vorhandenen Deklarationen. Die `#if`-Direktive dient dazu, die zweite Deklaration vor dem Compiler zu verbergen, lässt sie jedoch für den SQL-Präprozessor sichtbar.

Mehrere Deklarationen können nützlich sein, doch Sie müssen sicherstellen, dass jede Deklaration den einzelnen Typen dieselben Variablenamen gibt. Der Präprozessor geht davon aus, dass jede Hostvariable im Anschluss an ihre Deklaration global bekannt ist, da er die Sprache C nicht vollständig syntaktisch analysieren kann.

Indikatorvariablen

Indikatorvariablen sind C-Variablen, die zusätzliche Informationen über eine bestimmte Hostvariable enthalten. Sie können eine Hostvariable verwenden, um Daten abzurufen oder einzugeben. Mit Indikatorvariablen können Sie NULL bearbeiten.

Eine Indikatorvariable ist eine Hostvariable vom Typ a_sql_len, die in einer SQL-Anweisung unmittelbar auf eine normale Hostvariable folgt. Um NULL zu entdecken oder festzulegen, positionieren Sie die Indikatorvariable unmittelbar nach einer normalen Hostvariablen in einer SQL-Anweisung.

Beispiel

In folgender INSERT-Anweisung ist zum Beispiel **:ind_phone** eine Indikatorvariable.

```
EXEC SQL INSERT INTO Employee
   VALUES (:employee_number, :employee_name,
   :employee_initials, :employee_phone:ind_phone );
```

Wenn eine Fetch- oder Execute-Anweisung keine Zeilen vom Server erhält (wie z.B. bei einem Fehler oder wenn das Ende der Ergebnismenge erreicht ist), bleiben die Indikatorwerte unverändert.

Hinweis
Um in Zukunft die Verwendung von 32- und 64-Bit-Längen und -Indikatoren zu ermöglichen, wird 'short int' für Embedded SQL-Indikatorvariablen nicht mehr unterstützt. Verwenden Sie stattdessen a_sql_len.

Werte von Indikatorvariablen

Folgende Tabelle bietet eine Zusammenfassung zum Gebrauch von Indikatorvariablen:

Indikatorwert	Wert an die Datenbank übergeben	Wert von der Datenbank erhalten
0	Wert der Hostvariablen	Ruft einen Nicht-NULL-Wert ab
-1	NULL	Ruft NULL ab

Indikatorvariablen für die Behandlung von NULL

Das SQL-Konzept von NULL darf nicht mit der C-Sprachkonstanten mit demselben Namen verwechselt werden. In der SQL-Sprache repräsentiert NULL entweder ein unbekanntes Attribut oder eine ungeeignete Information. Die C-Sprachkonstante steht für einen Zeigerwert, der nicht auf eine Speicherposition zeigt.

Wenn NULL in der Dokumentation von SQL Anywhere verwendet wird, ist die oben genannte Bedeutung der SQL-Datenbank gemeint. Die gleichnamige C-Konstante wird dagegen als Null-Zeiger bezeichnet (Kleinschreibung).

NULL ist nicht dasselbe wie ein beliebiger anderer Wert des für die Spalte festgelegten Typs. Indikatorvariablen werden benötigt, NULL-Werte an die Datenbank zu übergeben oder NULL-Ergebnisse von ihr zu erhalten.

Indikatorvariablen verwenden, um NULL einzufügen

Eine INSERT-Anweisung kann eine Indikatorvariable wie folgt einbeziehen:

```
EXEC SQL BEGIN DECLARE SECTION;
short int employee_number;
char employee_name[50];
char employee_initials[6];
char employee_phone[15];
a_sql_len ind_phone;
EXEC SQL END DECLARE SECTION;
/* set values of employee number, name,
   initials, and phone number */
if( /* phone number is known */ ) {
   ind_phone =  0;
} else {
   ind_phone = -1; /* NULL */
}
EXEC SQL INSERT INTO Employee
   VALUES (:employee_number, :employee_name,
   :employee_initials, :employee_phone:ind_phone );
```

Hat die Indikatorvariable den Wert –1, wird NULL eingefügt. Hat sie den Wert 0, wird der tatsächliche Wert von employee_phone eingefügt.

Indikatorvariablen verwenden, um NULL abzurufen

Indikatorvariablen werden auch benutzt, um Daten von der Datenbank abzurufen. Sie zeigen an, dass der Wert NULL abgerufen wurde (Indikator ist negativ). Wenn der Wert NULL aus der Datenbank abgerufen wurde und keine Indikatorvariable verwendet wird, wird der Fehler SQLE_NO_INDICATOR generiert.

Siehe auch

- „SQL Communications Area initialisieren“ auf Seite 39

Abrufen von Daten

In Embedded SQL werden Daten mit der Anweisung SELECT abgerufen. Dabei werden zwei Fälle unterschieden:

1. Die SELECT-Anweisung gibt keine Zeilen oder genau eine Zeile zurück.
2. Die SELECT-Anweisung gibt mehrere Zeilen zurück.

Einzeiliges Abrufen

Eine einzeilige Abfrage fragt höchstens eine Zeile von der Datenbank ab. Eine SELECT-Anweisung für eine einzeilige Abfrage kann eine INTO-Klausel nach der Auswahlliste und vor der FROM-Klausel enthalten. Die INTO-Klausel enthält eine Liste der Hostvariablen, um den Wert der einzelnen Elemente der Auswahlliste zu empfangen. Die Anzahl der Hostvariablen muss mit der Anzahl der Auswahllisten-Elemente übereinstimmen. Die Hostvariablen können von Indikatorvariablen gefolgt sein, um NULL-Ergebnisse anzuzeigen.

Sobald die SELECT-Anweisung ausgeführt wird, ruft der Datenbankserver die Ergebnisse ab und schreibt sie in die Hostvariablen.

- Wenn die Abfrage mehrere Zeilen zurückgibt, gibt der Datenbankserver den Fehler SQLE_TOO_MANY_RECORDS zurück.
- Wenn die Abfrage keine Zeilen zurückgibt, wird die Warnung SQLE_NOTFOUND zurückgegeben.

Siehe auch

- „SQL Communications Area initialisieren“ auf Seite 39

Beispiel

Folgendes Codefragment gibt zum Beispiel 1 zurück, wenn eine Zeile der Tabelle employee erfolgreich abgerufen wird, 0, falls die Zeile nicht vorhanden ist, und -1, falls ein Fehler auftritt.

```
EXEC SQL BEGIN DECLARE SECTION;
   long int    emp_id;
   char        name[41];
   char        sex;
   char        birthdate[15];
   a_sql_len   ind_birthdate;
EXEC SQL END DECLARE SECTION;
int find_employee( long employee )
{
   emp_id = employee;
   EXEC SQL SELECT emp_fname || ' ' || emp_lname,
         sex, birth_date
      INTO :name, :sex, birthdate:ind_birthdate
      FROM "DBA".employee
      WHERE emp_id = :emp_id;
   if( SQLCODE == SQLE_NOTFOUND ) {
      return( 0 ); /* employee not found */
   } else if( SQLCODE < 0 ) {
      return( -1 ); /* error */
   } else {
      return( 1 ); /* found */
   }
}
```

Mehrere Zeilen abrufen

Verwenden Sie Cursor, um Zeilen aus einer Abfrage abzurufen, deren Ergebnismenge mehrere Zeilen umfasst. Ein Cursor ist ein Handle oder ein Bezeichner für die SQL-Abfrage und eine Position innerhalb der Ergebnismenge.

Cursor in UltraLite-Anwendungen werden immer mit der Option WITH HOLD geöffnet. Sie werden nie automatisch geschlossen. Sie müssen daher jeden Cursor explizit mit der Anweisung CLOSE schließen.

Ein Cursor kann mit den folgenden Schritten verwaltet werden:

1. Deklarieren Sie einen Cursor für eine bestimmte SELECT-Anweisung mit der DECLARE-Anweisung.
2. Öffnen Sie den Cursor mit der Anweisung OPEN.
3. Rufen Sie mit der Anweisung FETCH einzelne Zeilen aus dem Cursor ab.
4. Wiederholen Sie das Abrufen von Zeilen, bis die Warnung SQLE_NOTFOUND zurückgegeben wird. Fehler- und Warnungscodes werden in der Variable SQLCODE zurückgegeben, die im SQL-Kommunikationsbereich definiert wird.
5. Schließen Sie den Cursor mit der CLOSE-Anweisung.

Das folgende einfache Beispiel zeigt den Gebrauch von Cursor:

```
void print_employees( void )
{
   int status;
   EXEC SQL BEGIN DECLARE SECTION;
   char name[50];
   char sex;
   char birthdate[15];
   a_sql_len   ind_birthdate;
   EXEC SQL END DECLARE SECTION;
   /* 1. Declare the cursor. */
   EXEC SQL DECLARE C1 CURSOR FOR
      SELECT emp_fname || ' ' || emp_lname,
             sex, birth_date
      FROM "DBA".employee
      ORDER BY emp_fname, emp_lname;
   /* 2. Open the cursor. */
   EXEC SQL OPEN C1;
   /* 3. Fetch each row from the cursor. */
   for( ;; ) {
      EXEC SQL FETCH C1 INTO :name, :sex,
            :birthdate:ind_birthdate;
      if( SQLCODE == SQLE_NOTFOUND ) {
         break; /* no more rows */
      } else if( SQLCODE < 0 ) {
         break; /* the FETCH caused an error */
      }
      if( ind_birthdate < 0 ) {
         strcpy( birthdate, "UNKNOWN" );
      }
      printf( "Name: %s Sex: %c Birthdate:
               %s\n",name, sex, birthdate );
   }
```

```
    /* 4. Close the cursor. */
    EXEC SQL CLOSE C1;
}
```

Cursor positionieren

Ein Cursor wird an einer von drei Stellen positioniert:

- Auf einer Zeile
- Vor der ersten Zeile
- Nach der letzten Zeile

Reihenfolge der Zeilen in einem Cursor

Sie können die Reihenfolge der Zeilen in einem Cursor steuern, indem Sie in den SELECT-Anweisungen, die den betreffenden Cursor definieren, eine ORDER BY-Klausel einbeziehen. Wenn Sie diese Klausel nicht angeben, ist die Reihenfolge der Zeilen willkürlich.

Wenn Sie keine explizite Reihenfolge festlegen, ist nur sichergestellt, dass bei einem wiederholten Abrufen jede Zeile in der Ergebnismenge ein einziges Mal enthalten ist, bevor SQLE_NOTFOUND zurückgegeben wird.

Cursor neu positionieren

Wenn Sie einen Cursor öffnen, befindet er sich vor der ersten Zeile. Die Anweisung FETCH verschiebt die Cursorposition automatisch nach vorne. Ein Versuch, nach der letzten Zeile eine FETCH-Anweisung auszuführen, führt zu dem Fehler SQLE_NOTFOUND, der als praktisches Signal verwendet werden kann, um die sequenzielle Verarbeitung der Zeilen abzuschließen.

Sie können den Cursor auch an eine neue absolute Position und relativ zum Anfang oder zum Ende der Abfrageergebnisse positionieren oder ihn relativ zur aktuellen Cursorposition verschieben. Mit speziellen *positionierten* Versionen der Anweisungen UPDATE und DELETE können Sie die Zeile an der aktuellen Cursorposition aktualisieren oder löschen. Wenn sich der Cursor vor der ersten oder nach der letzten Zeile befindet, wird ein SQLE_NOTFOUND-Fehler zurückgegeben.

Um unvorhersehbare Ergebnisse bei der expliziten Positionierung zu vermeiden, können Sie eine ORDER BY-Klausel in die SELECT-Anweisung einbeziehen, die den Cursor festlegt.

Mit der Anweisung PUT können Sie eine neue Zeile in einen Cursor einfügen.

Cursorpositionierung nach Aktualisierung

Nach dem Aktualisieren von Daten, auf die ein offener Cursor zugreift, sollten die betreffenden Zeilen erneut abgerufen und angezeigt werden. Wird der Cursor verwendet, um eine einzelen Zeile anzuzeigen, wird mit FETCH RELATIVE 0 wieder die aktuelle Zeile gefangen. Wurde die aktuelle Zeile gelöscht, fängt der Cursor die nächste Zeile (oder es wird SQLE_NOTFOUND zurückgegeben, wenn es keine weitere Zeile mehr gibt).

Wenn eine temporäre Tabelle für den Cursor verwendet wird, erscheinen die in die zugrunde liegenden Tabellen eingefügten Zeilen nicht, bis der Cursor geschlossen und erneut geöffnet wird. Es ist schwierig zu beurteilen, ob eine temporäre Tabelle von einer SELECT-Anweisung betroffen ist, ohne den vom SQL-Präprozessor generierten Code zu untersuchen bzw. ohne genaue Kenntnis der Bedingungen, unter denen temporäre Tabellen verwendet werden. Sie können temporäre Tabellen gewöhnlich vermeiden, indem Sie in der ORDER BY-Klausel einen Index für die Spalten haben.

Einfügungen, Aktualisierungen und Löschungen in nicht-temporären Tabellen können sich auf die Cursorposition auswirken. Da UltraLite Cursor-Zeilen jeweils einzeln materialisiert, (wenn keine temporäre Tabellen verwendet werden), können sich die Daten aus einer soeben eingefügten Zeile (oder das Fehlen von Daten aus einer soeben gelöschten Zeile) in nachfolgenden FETCH-Vorgängen auswirken. In einem einfachen Fall, in dem (Teile von) Zeilen aus einer einzelnen Tabelle ausgewählt werden, erscheint eine eingefügte oder geänderte Zeile in der Ergebnismenge des Cursors, wenn sie die Auswahlkriterien der SELECT-Anweisung erfüllt. Ebenso ist eine soeben gelöschte Zeile, die zuvor zur Ergebnismenge beigetragen hat, nicht länger enthalten.

Siehe auch

- „FETCH-Anweisung [ESQL] [SP]“ [*SQL Anywhere Server - SQL-Referenzhandbuch*]
- „Cursor-Grundsätze“ [*SQL Anywhere Server - Programmierung*]
- „Tipp: Arbeitstabellen in der Abfrageverarbeitung verwenden ("Alle Zeilen" als Optimierungsziel verwenden)“ [*SQL Anywhere Server - SQL-Benutzerhandbuch*]

Benutzerauthentifizierung

Ein umfassendes Beispiel finden Sie im Verzeichnis *%SQLANYSAMP16%\UltraLite\esqlauth*. Der nachstehende Code stammt aus *%SQLANYSAMP16%\UltraLite\esqlauth\sample.sqc*.

```
//embedded SQL
app() {
   ...
/* Declare fields */
   EXEC SQL BEGIN DECLARE SECTION;
      char uid[31];
      char pwd[31];
   EXEC SQL END DECLARE SECTION;
   db_init( &sqlca );
   ...
   EXEC SQL CONNECT "DBA" IDENTIFIED BY "sql";
   if( SQLCODE == SQLE_NOERROR ) {
      printf("Enter new user ID and password\n" );
      scanf( "%s %s", uid, pwd );
      ULGrantConnectTo( &sqlca,
         UL_TEXT( uid ), UL_TEXT( pwd ) );
      if( SQLCODE == SQLE_NOERROR ) {
         // new user added: remove DBA
         ULRevokeConnectFrom( &sqlca, UL_TEXT("DBA") );
      }
      EXEC SQL DISCONNECT;
   }
   // Prompt for password
    printf("Enter user ID and password\n" );
    scanf( "%s %s", uid, pwd );
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
```

Der Code führt die folgenden Aufgaben aus:

1. Er initiiert die Datenbankfunktionalität durch Aufruf von db_init.
2. Er versucht, mit der Standardbenutzer-ID und dem Standardkennwort eine Verbindung herzustellen.
3. Er fügt einen neuen Benutzer hinzu, wenn der Verbindungsversuch erfolgreich ist.
4. Er löscht den Benutzer DBA aus der UltraLite-Datenbank, wenn der neue Benutzer erfolgreich hinzugefügt wurde.
5. Disconnect. Nun werden der Datenbank eine aktualisierte Benutzer-ID und ein aktualisiertes Kennwort hinzugefügt.
6. Mit der aktualisierten Benutzer-ID und dem aktualisierten Kennwort wird eine Verbindung hergestellt.

Siehe auch

- ULGrantConnectTo-Methode [UltraLite Embedded SQL] auf Seite 250
- ULRevokeConnectFrom-Methode [UltraLite Embedded SQL] auf Seite 253

Datenverschlüsselung mit UltraLite Embedded SQL

Sie können Ihre UltraLite-Datenbank verschlüsseln oder verschleiern, wenn Sie UltraLite Embedded SQL verwenden.

Verschlüsselung

Wenn eine UltraLite-Datenbank erstellt wird (z.B. mit Sybase Central), kann ein optionaler Chiffrierschlüssel angegeben werden. Der Chiffriercode wird verwendet, um die Datenbank zu verschlüsseln. Wenn die Datenbank verschlüsselt ist, muss bei allen nachfolgenden Verbindungsversuchen der Chiffrierschlüssel angegeben werden. Der angegebene Schlüssel wird mit dem Original-Chiffrierschlüssel verglichen. Wenn die Schlüssel nicht übereinstimmen, schlägt die Verbindung fehl.

Wählen Sie einen Wert für den Chiffrierschlüssel, der nicht einfach erraten werden kann. Der Schlüssel kann von beliebiger Länge sein, doch im Allgemeinen ist ein längerer Schlüssel besser, da ein kürzerer Schlüssel einfacher zu erraten ist. Das Kombinieren von Zahlen, Buchstaben und Sonderzeichen reduziert die Gefahr des Erratens.

Benutzen Sie in Ihrem Schlüssel kein Semikolon. Schließen Sie den Schlüssel nicht in Anführungszeichen ein, da die Anführungszeichen sonst als Teil des Schlüssels interpretiert werden.

Die folgende Vorgehensweise wird üblicherweise verwendet, um eine Verbindung mit einer verschlüsselten UltraLite-Datenbank herzustellen:

1. Geben Sie den Chiffrierschlüssel in der Verbindungszeichenfolge der EXEC SQL CONNECT-Anweisung an.

2. Der Chiffrierschlüssel wird im Verbindungszeichenfolgen-Parameter key= angegeben.

   Sie müssen diesen Schlüssel immer dann angeben, wenn Sie eine Verbindung zur Datenbank herstellen wollen. Wenn Sie den Schlüssel verlieren, können Sie nie mehr auf die Datenbank zugreifen.

3. Programmieren Sie eine Behandlungsroutine für den Fall, dass versucht wird, eine verschlüsselte Datenbank mit dem falschen Schlüssel zu öffnen.

   Wenn versucht wird, eine verschlüsselte Datenbank zu öffnen und dabei ein falscher Schlüssel übergeben wird, gibt db_init den Wert ul_false zurück und SQLCODE wird auf -840 gesetzt.

Chiffrierschlüssel ändern

Sie können den Chiffrierschlüssel einer Datenbank ändern. Die Anwendung muss bereits mit dem alten Chiffrierschlüssel mit der Datenbank verbunden sein, bevor diese Änderung durchgeführt wird.

Übergeben Sie den neuen Schlüssel als Argument an die ULChangeEncryptionKey-Methode.

Verschleierung

Alternativ zur Datenbankverschlüsselung können Sie die Verschleierung der Datenbank festlegen. Die Verschleierung ist eine einfache Maskierung der Daten in der Datenbank, mit der verhindert werden soll,

dass die Daten in der Datenbank mit einem einfachen Anzeigeprogramm durchsucht werden können. Die Verschleierung ist eine Option, die zum Zeitpunkt der Datenbankerstellung angegeben wird.

Siehe auch

- ULChangeEncryptionKey-Methode [UltraLite Embedded SQL] auf Seite 233
- „UltraLiteDatenbank-Erstellungsparameter“ [*UltraLite - Datenbankverwaltung*]

Synchronisation für eine Embedded SQL-Anwendung einrichten

Bei der Synchronisation handelt es sich um eine der wichtigsten Funktionen vieler UltraLite-Anwendungen. Dieser Abschnitt beschreibt, wie Sie Ihrer Anwendung eine Synchronisationsfunktion hinzufügen.

Mitglieder der Strukturen in der Embedded SQL-API haben Ähnlichkeiten zur UltraLite C++-API.

Die Synchronisationslogik, die UltraLite-Anwendungen mit der konsolidierten Datenbank abstimmt, ist nicht in der Anwendung selbst gespeichert. Synchronisationsskripten in der konsolidierten Datenbank steuern zusammen mit dem MobiLink-Server und der UltraLite-Laufzeitbibliothek, wie Änderungen beim Senden verarbeitet werden und welche Änderungen eingelesen werden.

Die Details der einzelnen Synchronisationen werden von einer Reihe von Synchronisationsparametern gesteuert. Diese Parameter werden in einer Struktur gesammelt, die dann in einem Methodenaufruf für die Synchronisation als Argument angegeben wird. Die Methode ist bei allen Entwicklungsmodellen gleich.

Die folgende Vorgehensweise wird üblicherweise verwendet, um Ihrer Anwendung Synchronisation hinzuzufügen:

1. Initialisieren Sie die Struktur, die die Synchronisationsparameter enthält.
2. Weisen Sie die Parameterwerte für Ihre Anwendung zu.
3. Rufen Sie die Synchronisationsmethode auf und geben Sie die Struktur bzw. das Objekt als Argument an.

   Stellen Sie sicher, dass keine noch nicht festgeschriebenen Änderungen vorhanden sind, wenn Sie synchronisieren.

Siehe auch

- „Synchronisationsparameter initialisieren“ auf Seite 59
- „UltraLite-Netzwerkprotokolloptionen für dbmlsync“ [*UltraLite - Datenbankverwaltung*]
- „Synchronisationsaufruf“ auf Seite 59
- ul_sync_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 96
- „UltraLite-Synchronisationsparameter“ [*UltraLite - Datenbankverwaltung*]

Synchronisationsparameter initialisieren

Die Synchronisationsparameter werden in einer ul_sync_info-Struktur gespeichert.

Die Mitglieder der ul_sync_info-Struktur sind bei der Initialisierung nicht festgelegt. Daher müssen Sie Ihre Parameter mit dem Aufruf einer speziellen Methode auf ihren Ausgangswert setzen. Die Synchronisationsparameter werden in einer Struktur in der UltraLite-Header-Datei *%SQLANY16%\SDK\Include\ulglobal.h* festgelegt.

Siehe auch

- ULInitSyncInfo-Methode [UltraLite Embedded SQL] auf Seite 250

Beispiel

Das folgende Beispiel zeigt, wie Synchronisationsparameter mit der ULInitSyncInfo-Methode initialisiert werden:

```
ul_sync_info synch_info;
ULInitSyncInfo( &synch_info );
```

Beispiel

Der nachstehende Code leitet die TCP/IP-Synchronisation ein. Der MobiLink-Benutzername ist `Betty Best`, das Kennwort ist `TwentyFour`, die Skriptversion ist `default` und der MobiLink-Server läuft auf dem Hostcomputer `test.internal` auf Port `2439`:

```
ul_sync_info synch_info;
ULInitSyncInfo( &synch_info );
synch_info.user_name = UL_TEXT("Betty Best");
synch_info.password = UL_TEXT("TwentyFour");
synch_info.version = UL_TEXT("default");
synch_info.stream = ULSocketStream();
synch_info.stream_parms =
   UL_TEXT("host=test.internal;port=2439");
ULSynchronize( &sqlca, &synch_info );
```

Synchronisationsaufruf

Wie die Synchronisation im Einzelnen aufgerufen wird, hängt von der Zielplattform und vom Synchronisationsdatenstrom ab.

Der Synchronisationsprozess kann nur korrekt ablaufen, wenn das Gerät, auf dem die UltraLite-Anwendung ausgeführt wird, mit dem MobiLink-Server kommunizieren kann. Auf einigen Plattformen muss das Gerät physisch verbunden sein, indem Sie es in seine Dockingstation setzen oder mit einem geeigneten Kabel an einen Server anschließen. Wenn die Synchronisation nicht abgeschlossen werden kann, fügen Sie Code für die Fehlerbehandlung in Ihrer Anwendung hinzu.

Um die Synchronisation aufzurufen, rufen Sie die ULInitSyncInfo-Methode auf, um die Synchronisationsparameter zu initialisieren. Rufen Sie anschließend die ULSynchronize-Methode auf, um die Synchronisation durchzuführen.

Der Synchronisationsaufruf setzt eine Struktur voraus, die eine Gruppe von Parametern enthalten kann, die die Details der Synchronisation beschreiben. Die einzelnen Parameter sind vom jeweiligen Datenstrom abhängig.

Festgeschriebene Änderungen und Synchronisation

Eine UltraLite-Datenbank darf keine noch nicht festgeschriebenen Änderungen enthalten, wenn sie synchronisiert wird. Wenn Sie versuchen, eine UltraLite-Datenbank zu synchronisieren, wenn eine Verbindung eine noch nicht festgeschriebene Transaktion enthält, schlägt die Synchronisation fehl. Außerdem wird eine Ausnahmebedingung erzeugt und der Fehler SQLE_UNCOMMITTED_TRANSACTIONS protokolliert. Dieser Fehlercode erscheint auch im MobiLink-Serverlog.

Siehe auch

- „Synchronisationsparameter Download Only“ [*UltraLite - Datenbankverwaltung*]

Ausgangsdaten für Ihre Anwendung

Viele UltraLite-Anwendungen benötigen Daten, um damit arbeiten zu können. Sie können Daten in Ihre Anwendung einlesen, indem Sie sie synchronisieren. Sie sollten in Ihre Anwendung eine entsprechende Logik aufnehmen, um sicherzustellen, dass bei ihrem ersten Aufruf und vor allen anderen Aktionen alle erforderlichen Daten eingelesen werden.

Tipp
Es ist einfacher, Fehler zu finden, wenn Sie eine Anwendung in verschiedenen Stufen entwickeln. Bei der Entwicklung eines Prototyps sollten Sie vorübergehend INSERT-Anweisungen in Ihrer Anwendung verwenden, um Daten für Test- und Vorführzwecke bereitzustellen. Wenn Ihr Prototyp richtig arbeitet, ersetzen Sie die termporären INSERT-Anweisungen durch den Code, um die Synchronisation durchzuführen.

Siehe auch

- „MobiLink-Entwicklungstipps“ [*MobiLink - Serveradministration*]

Kommunikationsfehler während der Synchronisation

Der folgende Code veranschaulicht, wie Kommunikationsfehler aus Embedded SQL-Anweisungen verarbeitet werden.

```
if( psqlca->sqlcode == SQLE_MOBILINK_COMMUNICATIONS_ERROR ) {
     printf( "  Stream error information:\n"
             "    stream_error_code = %ld\t(ss_error_code)\n"
             "    error_string      = \"%s\"\n"
             "    system_error_code = %ld\n",
     (long)info.stream_error.stream_error_code,
     info.stream_error.error_string,
     (long)info.stream_error.system_error_code );
}
```

SQLE_MOBILINK_COMMUNICATIONS_ERROR ist der allgemeine Fehlercode für Kommunikationsfehler.

Um UltraLite klein zu halten, übergibt die Runtime-Engine Zahlen anstelle von Meldungen.

Überwachen und Abbrechen der Synchronisation

Dieser Abschnitt beschreibt, wie Sie die Synchronisation von UltraLite-Anwendungen aus überwachen und abbrechen können.

Synchronisation überwachen

- Geben Sie im Mitglied observer in der Synchronisationsstruktur (ul_synch_info) den Namen Ihrer Callback-Funktion an.
- Rufen Sie die Synchronisationsfunktion bzw. Synchronisationsmethode auf, um die Synchronisation zu starten.
- UltraLite ruft Ihre Callback-Funktion immer dann auf, wenn der Synchronisationsstatus sich ändert. Im folgenden Abschnitt wird der Synchronisationsstatus beschrieben.

Folgender Code zeigt, wie diese Task-Sequenz in einer Embedded SQL-Anwendung implementiert werden kann:

```
ULInitSyncInfo( &info );
info.user_name = m_EmpIDStr;
...
//The info parameter of ULSynchronize() contains
// a pointer to the observer function
info.observer = ObserverFunc;
ULSynchronize( &sqlca, &info );
```

Informationen zum Synchronisationsstatus

Die Callback-Funktion, mit der Sie die Synchronisation überwachen, verwendet die ul_sync_status-Struktur als Parameter.

Die Struktur ul_sync_status hat folgende Mitglieder:

```
struct ul_sync_status {
    struct {
       ul_u_long   bytes;
       ul_u_long   inserts;
       ul_u_long   updates;
       ul_u_long   deletes;
     }               sent;
    struct {
       ul_u_long   bytes;
       ul_u_long   inserts;
       ul_u_long   updates;
       ul_u_long   deletes;
     }               received;
    p_ul_sync_info  info;
    ul_sync_state   state;
    ul_u_short       db_table_count;
```

```
    ul_u_short      table_id;
    char            table_name[];
    ul_wchar        table_name_w2[];
    ul_u_short      sync_table_count;
    ul_u_short      sync_table_index;
    ul_sync_state   state;
    ul_bool         stop;
    ul_u_short      flags;
    ul_void *       user_data;
    SQLCA *         sqlca;
    ul_u_long       current_download_row_count;
    ul_u_long       total_download_row_count;
}
```

- **sent.inserts** Die Anzahl der bisher eingefügten heraufgeladenen Zeilen.
- **sent.updates** Die Anzahl der bisher aktualisierten heraufgeladenen Zeilen.
- **sent.deletes** Die Anzahl der bisher gelöschten heraufgeladenen Zeilen.
- **sent.bytes** Die Anzahl der Byte, die bisher heraufgeladen wurden.
- **received.inserts** Die Anzahl der bisher eingefügten heruntergeladenen Zeilen.
- **received.updates** Die Anzahl der bisher aktualisierten heruntergeladenen Zeilen.
- **received.deletes** Die Anzahl der bis dahin gelöschten heruntergeladenen Zeilen.
- **received.bytes** Die Anzahl der Bytes, die bisher heruntergeladen wurden.
- **info** Gibt einen Zeiger auf die ul_sync_info-Struktur zurück.
- **db_table_count** Gibt die Anzahl der Tabellen in der Datenbank zurück.
- **table_id** Gibt die aktuelle Tabellennummer (in Bezug auf 1) der im Upload oder Download begriffenen Tabelle zurück. Bei dieser Nummer werden Werte übersprungen, wenn nicht alle Tabellen synchronisiert werden. Außerdem wird diese Nummer nicht unbedingt hochgezählt.
- **table_name[]** Gibt den Namen der aktuellen Tabelle zurück.
- **table_name_w2[]** Gibt den Namen der aktuellen Tabelle zurück (Version mit breiten Zeichen). Dieses Feld wird nur unter Windows (PC und Mobile) gefüllt.
- **sync_table_count** Gibt die Anzahl der Tabellen an, die synchronisiert werden.
- **sync_table_index** Gibt die Nummer der Tabelle im Upload oder Download zurück, beginnend mit 1 und endend mit dem **sync_table_count**-Wert. Bei dieser Nummer werden Werte übersprungen, wenn nicht alle Tabellen synchronisiert werden.
- **state** Einer der folgenden Statuswerte:
  - **UL_SYNC_STATE_STARTING** Es wurden noch keine Synchronisationsaktionen ausgeführt.
  - **UL_SYNC_STATE_CONNECTING** Der Synchronisationsdatenstrom wurde erstellt, aber noch nicht geöffnet.

  - **UL_SYNC_STATE_SENDING_HEADER** Der Synchronisationsdatenstrom wurde geöffnet und der Header wird demnächst gesendet.
  - **UL_SYNC_STATE_SENDING_TABLE** Es wird gerade eine Tabelle gesendet.
  - **UL_SYNC_STATE_SENDING_DATA** Schemainformationen oder Daten werden gesendet.
  - **UL_SYNC_STATE_FINISHING_UPLOAD** Die Upload-Phase ist abgeschlossen und eine Festschreibung wird ausgeführt.
  - **UL_SYNC_STATE_RECEIVING_UPLOAD_ACK** Es wird gerade eine Bestätigung darüber empfangen, dass der Sendevorgang abgeschlossen wurde.
  - **UL_SYNC_STATE_RECEIVING_TABLE** Es wird gerade eine Tabelle empfangen.
  - **UL_SYNC_STATE_RECEIVING_DATA** Schemainformationen oder Daten werden empfangen.
  - **UL_SYNC_STATE_COMMITING_DOWNLOAD** Die Download-Phase ist beendet und eine Festschreibung wird ausgeführt.
  - **UL_SYNC_STATE_SENDING_DOWNLOAD_ACK** Es wird gerade eine Bestätigung gesendet, dass der Download-Vorgang abgeschlossen wurde.
  - **UL_SYNC_STATE_DISCONNECTING** Der Synchronisationsdatenstrom wird demnächst geschlossen.
  - **UL_SYNC_STATE_DONE** Die Synchronisation wurde erfolgreich abgeschlossen.
  - **UL_SYNC_STATE_ERROR** Die Synchronisation wurde abgeschlossen, jedoch mit einem Fehler.
  - **UL_SYNC_STATE_ROLLING_BACK_DOWNLOAD** Während des Downloads ist ein Fehler aufgetreten und der Download wird zurückgesetzt.

- **stop** Setzen Sie dieses Mitglied auf TRUE, um die Synchronisation zu unterbrechen. Die SQL-Ausnahmebedingung SQLE_INTERRUPTED wird gesetzt, und die Synchronisation wird gestoppt, als ob ein Kommunikationsfehler aufgetreten wäre. Der Beobachter wird *immer* entweder mit dem Zustand DONE oder ERROR aufgerufen, sodass er richtig aufräumen kann.
- **flags** Gibt die aktuellen Optionen für die Synchronisation zurück, die weitere Informationen über den aktuellen Status liefern.
- **user_data** Gibt das Benutzerdatenobjekt zurück, das als Argument an die ULSetSynchronizationCallback-Funktion übergeben wird.
- **sqlca** Gibt den Zeiger auf den aktiven SQLCA-Bereich der Verbindung zurück.
- **current_download_row_count** Gibt die Anzahl der bisher heruntergeladenen Zeilen zurück. Diese Anzahl umfasst Duplikatzeilen, die nicht in received.inserts, received.updates oder received.deletes enthalten sind.

- **total_download_row_count** Gibt die Gesamtzahl der in dem Download zu empfangenden Zeilen zurück. Diese Anzahl umfasst Duplikatzeilen, die nicht in received.inserts, received.updates oder received.deletes enthalten sind.

Beispiel

Der folgende Code zeigt eine einfache Beobachtungsfunktion:

```
extern void __stdcall ObserverFunc(
    p_ul_sync_status status )
{
    switch( status->state ) {
        case UL_SYNC_STATE_STARTING:
        printf( "Starting\n");
        break;
        case UL_SYNC_STATE_CONNECTING:
            printf( "Connecting\n"  );
            break;
        case UL_SYNC_STATE_SENDING_HEADER:
            printf( "Sending Header\n" );
            break;
        case UL_SYNC_STATE_SENDING_TABLE:
            printf( "Sending Table %d of %d\n",
                    status->tableIndex + 1,
                    status->tableCount );
            break;
        case UL_SYNCH_RECEIVING_UPLOAD_ACK:
            printf( "Receiving Upload Ack\n" );
            break;
        case UL_SYNC_STATE_RECEIVING_TABLE:
            printf( "Receiving Table %d of %d\n",
                    status->tableIndex + 1,
                    status->tableCount );
            break;
        case UL_SYNC_STATE_SENDING_DOWNLOAD_ACK:
            printf( "Sending Download Ack\n" );
            break;
        case UL_SYNC_STATE_DISCONNECTING:
            printf( "Disconnecting\n" );
            break;
        case UL_SYNC_STATE_DONE:
            printf( "Done\n" );
            break;
        break;
...
```

Dieser Beobachter erzeugt bei der Synchronisation von zwei Tabellen die folgende Ausgabe:

```
Starting
Connecting
Sending Header
Sending Table 1 of 2
Sending Table 2 of 2
Receiving Upload Ack
Receiving Table 1 of 2
Receiving Table 2 of 2
Sending Download Ack
Disconnecting
Done
```

CustDB-Beispiel

Die Beispielanwendung CustDB enthält ein Beispiel einer Beobachtungsfunktion. Die Implementierung in CustDB umfasst ein Fenster, das den Verarbeitungsfortschritt der Synchronisation anzeigt und dem Benutzer gestattet, die Synchronisation abzubrechen. Durch die Komponente für die Benutzeroberfläche wird die Beobachtungsfunktion plattformspezifisch.

Der CustDB-Beispielcode befindet sich im Verzeichnis *%SQLANYSAMP16%\UltraLite\CustDB*. Die Beobachtungsfunktion befindet sich in den plattformspezifischen Unterverzeichnissen des *CustDB*-Verzeichnisses.

Siehe auch

- ul_sync_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 96
- ul_sync_status-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 101
- „Der Synchronisationsprozess“ [*MobiLink - Erste Orientierung*]

Erstellen von Embedded SQL-Anwendungen

In diesem Abschnitt wird der allgemeine Erstellungsvorgang für Embedded SQL-Anwendungen in UltraLite erläutert.

Bevor Sie diesen Abschnitt lesen, sollten Sie sich mit dem Modell der Embedded SQL-Entwicklung vertraut machen.

Allgemeine Aufbauprozeduren

Die folgende Vorgehensweise wird üblicherweise verwendet, wenn Sie eine Embedded SQL-Anwendung für UltraLite aufbauen:

1. Lassen Sie den SQL-Präprozessor über *jede* der Quelldateien in Embedded SQL laufen.

   Der SQL-Präprozessor ist das sqlpp-Befehlszeilen-Dienstprogramm. Er erledigt die Vorverarbeitung der Embedded SQL-Quelldateien und produziert C-Quelldateien, die in Ihre Anwendung kompiliert werden müssen.

   Vorsicht
   sqlpp überschreibt die Ausgabedatei unabhängig von ihrem Inhalt. Stellen Sie sicher, dass der Name der Ausgabedatei nicht mit dem Namen einer Ihrer Quelldateien übereinstimmt. Standardmäßig ändert sqlpp den Namen der Ausgabedatei, indem es das Suffix Ihrer Quelldatei in *.cpp* ändert. Bei Zweifeln geben Sie den Namen der Ausgabedatei hinter dem Namen der Quelldatei explizit an.

2. Kompilieren Sie *alle* C++-Quelldateien für die Zielplattform Ihrer Wahl. Einbeziehen:

   - Alle vom SQL-Präprozessor generierten C++-Dateien
   - Alle weiteren C- oder C++-Quelldateien, die von der Anwendung benötigt werden

3. Verknüpfen Sie *alle* diese Objektdateien mit der UltraLite-Laufzeitbibliothek.

Eine makefile-Datei, die diesen Prozess verwendet, befindet sich im Verzeichnis *%SQLANYSAMP16%\UltraLite\ESQLSecurity*.

Hinweis
Erforderliche getrennt lizenzierbare Komponenten.

FIPS-zertifizierte Verschlüsselung erfordert eine separate Lizenz. Alle Technologien für starke Verschlüsselungen unterliegen Exportbestimmungen.

Siehe „Getrennt lizenzierbare Komponenten“ [*SQL Anywhere 16 - Einführung*].

Siehe auch

- „UltraLite-Dienstprogramm für den SQL-Präprozessor (sqlpp)“ [*UltraLite - Datenbankverwaltung*]

Entwicklungs-Tool-Konfiguration für die Embedded SQL-Entwicklung

Viele Entwicklungstools verwenden ein Abhängigkeitsmodell, bei dem es sich um eine Make-Datei handeln kann, in der der Zeitstempel der einzelnen Quelldateien mit dem der Zieldateien (meist Objektdateien) verglichen wird, um zu bestimmen, ob die Zieldatei neu generiert werden muss.

Bei der UltraLite-Entwicklung bedeutet eine Änderung einer SQL-Anweisung in einem Entwicklungsprojekt, dass der erzeugte Code neu generiert werden muss. Änderungen werden nicht im Zeitstempel einer einzelnen Quelldatei widergespiegelt, da die SQL-Anweisungen in der Referenzdatenbank gespeichert werden.

Siehe auch

- „UltraLite-Dienstprogramm für den SQL-Präprozessor (sqlpp)“ [*UltraLite - Datenbankverwaltung*]

SQL-Präprozessor ausführen

Integrieren Sie den SQL-Präprozessor in eine abhängigkeitsbasierte Entwicklungsumgebung, indem Sie Anweisungen für die Ausführung mit Visual C++ hinzufügen.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Fügen Sie Ihrem Entwicklungsprojekt die *.sqc*-Dateien hinzu.

   Das Entwicklungsprojekt wird in Ihrem Entwicklungstool festgelegt.

2. Fügen Sie jeder *.sqc*-Datei eine benutzerdefinierte Erstellungsregel hinzu.

- Die benutzerdefinierte Erstellungsregel sollte den SQL-Präprozessor starten. In Visual C++ muss die Erstellungsregel folgenden Befehl enthalten (einzugeben in einer einzelnen Zeile):

  ```
  "%SQLANY16%\Bin32\sqlpp.exe" -q -u $(InputPath) $(InputName).cpp
  ```

  Dabei gilt: SQLANY16 ist eine Umgebungsvariable, die auf Ihr SQL Anywhere-Installationsverzeichnis zeigt.
- Stellen Sie die Ausgabe für den Befehl auf **$(InputName).cpp** ein.

3. Kompilieren Sie die *.sqc*-Dateien und fügen Sie dem Entwicklungsprojekt die generierten *.cpp*-Dateien hinzu.

   Sie müssen die generierten Dateien in Ihr Projekt einfügen, auch wenn es sich dabei nicht um die Quelldateien handelt, damit Sie Abhängigkeiten einrichten und Optionen erstellen können.

4. Legen Sie für alle generierten *.cpp*-Dateien die Präprozessor-Definitionen fest.

   - Fügen Sie den Präprozessor-Definitionen unter "Allgemein" oder "Präprozessor" die Angabe UL_USE_DLL hinzu.
   - Fügen Sie unter Präprozessor *$(SQLANY16)\SDK\Include* und alle anderen Include-Ordner, die Sie in Ihrem Include-Pfad wünschen, als kommagetrennte Liste ein.

Ergebnisse

Der SQL-Präprozessor wird für die Visual C++ Entwicklung konfiguriert.

UltraLite-Anwendungsentwicklung für Windows Mobile

Mit Microsoft Visual Studio 2005 können Anwendungen für die Windows Mobile-Umgebung entwickelt werden.

Anwendungen für Windows Mobile sollten die Standardeinstellung für wchar_t verwenden und mit den UltraLite-Laufzeitbibliotheken in *\Program Files\SQLAny16\ultralite\ce\arm.50\lib* verknüpfen.

Sie können Ihre Anwendungen auf den meisten Windows Mobile-Plattformen auf einem Emulator testen.

Siehe auch

- „Unterstützte Plattformen“ [*SQL Anywhere 16 - Einführung*]

CustDB-Beispielanwendung

CustDB ist eine einfache Anwendung zum Status von Kundenbestellungen, die als Visual Studio-Lösung bereitgestellt wird. Sie befindet sich im Verzeichnis *%SQLANYSAMP16%\UltraLite* Ihrer SQL Anywhere-Installation.

Hinweis
Das Beispielprojekt verwendet, soweit möglich, Umgebungsvariablen. Es kann erforderlich sein, das Projekt anzupassen, damit die Anwendung einwandfrei erstellt werden kann. Falls Sie auf Probleme stoßen, sollten Sie nach fehlenden Dateien in den Microsoft VC++-Ordnern suchen und die entsprechenden Verzeichniseinstellungen hinzufügen.

Bei Embedded SQL verwendet der Erstellungsprozess den SQL-Präprozessor sqlpp, um die Datei *CustDB.sqc* in die Datei *CustDB.cpp* zu verarbeiten. Dieser einstufige Prozess ist in kleineren UltraLite-Anwendungen nützlich, bei denen der gesamte Embedded SQL-Code auf ein Quellmodul begrenzt werden kann. In größeren UltraLite-Anwendungen müssen Sie mehrere sqlpp-Aufrufe verwenden.

Siehe auch

- „Die Beispieldatenbankanwendung CustDB“ [*SQL Anywhere 16 - Einführung*]
- „Erstellen von Embedded SQL-Anwendungen“ auf Seite 65

Beispielanwendung CustDB erstellen

Erstellen Sie die CustDB-Beispielanwendung, um herauszufinden, wie eine Anwendung mit einer UltraLite-Datenbank zusammenarbeitet.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Starten Sie Visual Studio.
2. Öffnen Sie die Projektdatei, die sich im Verzeichnis *%SQLANYSAMP16%\UltraLite\CustDB* befindet.
3. Klicken Sie auf **Build** » **Set Configuration Manager**, um die Zielplattform festzulegen.

   Wählen Sie eine aktive Lösungsplattform Ihrer Wahl aus.
4. Erstellen Sie die Anwendung:

   - Klicken Sie auf **Build** » **Projektmappe bereitstellen**, um die Anwendung CustDB zu erstellen und bereitzustellen.

     Nach der Erstellung der Anwendung wird sie automatisch auf das entfernte Gerät hochgeladen.
5. Starten Sie den MobiLink-Server.

   - Um den MobiLink-Server zu starten, klicken Sie auf **Start** » **Programme** » **SQL Anywhere 16** » **MobiLink** » **Synchronisationsserver-Beispiel**.
6. Führen Sie die Anwendung CustDB aus.

Vor dem Ausführen der CustDB-Anwendung muss die Datenbank custdb in das Stammverzeichnis des Geräts kopiert werden. Kopieren Sie die Datenbankdatei *%SQLANYSAMP16%\UltraLite\CustDB\custdb.udb* in das Stammverzeichnis des Geräts.

Auf dem Gerät oder Simulator führen Sie *CustDB.exe* im Projektordner unter *\Program Files* aus.

Ergebnisse

Die CustDB-Anwendung wird geladen.

Beständige Daten

Die UltraLite-Datenbank wird im Dateisystem von Windows Mobile gespeichert. Die Standarddatei ist *\UltraLiteDB\ul_store.udb*. Sie können diese Wahl mithilfe des Verbindungsparameters **file_name** überschreiben, mit dem der vollständige Pfadname des dateibasierten beständigen Speichers angegeben wird.

Die UltraLite-Laufzeitumgebung führt keine Ersetzungen des Parameters **file_name** aus. Wenn ein Verzeichnis erstellt werden muss, damit der Dateiname gültig ist, muss die Anwendung sicherstellen, dass alle Verzeichnisse vor dem Aufruf von **db_init** erstellt werden.

Sie können z.B. eine Flash-Speicherkarte verwenden, indem Sie nach Speicherkarten suchen und dem Namen den betreffenden Verzeichnisnamen für die Speicherkarte voranstellen. Beispiel:

```
file_name = "\\Storage Card\\My Documents\\flash.udb"
```

Klassennamen für Anwendungen zuordnen

Ordnen Sie Ihrer Anwendung einen eindeutigen Klassennamen zu, wenn Sie MFC benutzen. Beim Registrieren von Anwendungen für die Verwendung mit ActiveSync müssen Sie einen Fensterklassennamen angeben. Die Zuordnung von Klassennamen erfolgt während der Entwicklung und die Dokumentation für Ihr Anwendungsentwicklungstool ist die wichtigste Informationsquelle zu diesem Thema.

Voraussetzungen

Es gibt keine Voraussetzungen für diese Aufgabe.

Aufgabe

1. Erstellen und registrieren Sie eine benutzerdefinierte Fensterklasse für Dialogfelder auf der Basis der Standardklasse.

   Fügen Sie dem Startcode Ihrer Anwendung den folgenden Code hinzu. Der Code muss ausgeführt werden, bevor irgendwelche Dialoge erstellt werden.

   ```
   WNDCLASS wc;
   if( ! GetClassInfo( NULL, L"Dialog", &wc ) ) {
    AfxMessageBox( L"Error getting class info" );
   ```



```
}
wc.lpszClassName = L"MY_APP_CLASS";
if( ! AfxRegisterClass( &wc ) ) {
 AfxMessageBox( L"Error registering class" );
}
```

Dabei ist *MY_APP_CLASS* der eindeutige Klassenname für Ihre Anwendung.

2. Legen Sie fest, welches Dialogfeld das Hauptdialogfeld für Ihre Anwendung ist.

   Wenn Ihr Projekt mit dem MFC-Anwendungsassistenten erstellt wurde, ist dies wahrscheinlich ein Dialogfeld namens MyAppDlg.

3. Suchen Sie die Ressourcen-ID für das Hauptdialogfeld und schreiben Sie sie auf.

   Die Ressourcen-ID ist eine Konstante des gleichen allgemeinen Formats wie IDD_MYAPP_DIALOG.

4. Sorgen Sie dafür, dass das Hauptdialogfeld immer geöffnet ist, wenn Ihre Anwendung ausgeführt wird.

   Fügen Sie der InitInstance-Methode Ihrer Anwendung den folgenden Code hinzu.

   ```
   m_pMainWnd = &dlg;
   ```

   Durch diesen Code wird sichergestellt, dass beim Schließen des dlg main-Dialogs auch die Anwendung geschlossen wird. Weitere Hinweise finden Sie in der Microsoft-Dokumentation zu CWinThread::m_pMainWnd.

   Falls das Dialogfeld während der Ausführungszeit Ihrer Anwendung nicht geöffnet bleibt, müssen Sie auch die Fensterklasse der anderen Dialogfelder ändern.

5. Speichern Sie Ihre Änderungen.

6. Bearbeiten Sie die Ressourcendatei für Ihr Projekt.

   Öffnen Sie Ihre Ressourcendatei (welche die Erweiterung *.rc* hat) in einem Texteditor wie z.B. dem Windows-Editor.

7. Suchen Sie die Ressourcen-ID Ihres Hauptdialogfeldes.

   Ändern Sie die Definition des Hauptdialogfeldes für die Verwendung der neuen Fensterklasse wie im folgenden Beispiel. Die *einzige* Änderung, die Sie durchführen sollten, ist das Hinzufügen der Zeile **CLASS**:

   ```
   IDD_MYAPP_DIALOG DIALOG DISCARDABLE  0, 0, 139, 103
   STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION
   EXSTYLE WS_EX_APPWINDOW | WS_EX_CAPTIONOKBTN
   CAPTION "MyApp"
   FONT 8, "System"
   CLASS "MY_APP_CLASS"
   BEGIN
         LTEXT   "TODO: Place dialog controls here.",IDC_STATIC,13,33,112,17
   END
   ```

Dabei ist *MY_APP_CLASS* der Name der Fensterklasse, die Sie zuvor benutzt haben.

8. Speichern Sie die *.rc*-Datei.

9. Fügen Sie den Code zum Abfangen der Synchronisationsnachricht hinzu.

Ergebnisse

Eine eindeutiger Klassenname für Ihre Anwendung wird erstellt.

Siehe auch

- „ActiveSync-Synchronisation in der main dialog-Klasse hinzufügen“ auf Seite 72

Synchronisation unter Windows Mobile

UltraLite-Anwendungen unter Windows Mobile können über folgende Datenströme synchronisieren:

- ActiveSync
- TCP/IP
- HTTP

Die Parameter *user_name* und *stream_parms* müssen bei der Initialisierung innerhalb des Makros **UL_TEXT()** für Windows Mobile stehen, da die Kompilierungsumgebung breite Unicode-Zeichen verwendet.

Siehe auch

- „ActiveSync-Synchronisation einrichten“ auf Seite 71
- „TCP/IP-, HTTP-, oder HTTPS-Synchronisation unter Windows Mobile“ auf Seite 74
- „TCP/IP-, HTTP-, oder HTTPS-Synchronisation unter Windows Mobile“ auf Seite 74
- „UltraLite-Synchronisationsparameter“ [*UltraLite - Datenbankverwaltung*]

ActiveSync-Synchronisation einrichten

ActiveSync ist Software von Microsoft, die die Datensynchronisation zwischen einem PC mit Windows und einem verbundenen Windows Mobile-Handheld-Gerät abwickelt. UltraLite unterstützt die ActiveSync-Versionen 3.5 und später.

In diesem Abschnitt wird beschrieben, wie Ihrer Anwendung ActiveSync-Provider hinzugefügt wird und wie Ihre Anwendung für die Verwendung mit ActiveSync auf den Computern Ihrer Endbenutzer registriert wird.

Wenn Sie ActiveSync benutzen, kann die Synchronisation nur durch ActiveSync selbst initiiert werden. ActiveSync initiiert automatisch eine Synchronisation, wenn das Gerät in die Dockingstation gestellt oder der Befehl **Synchronisieren** im ActiveSync-Fenster gewählt wird. Der MobiLink-Provider startet die Anwendung, falls sie nicht bereits läuft, und sendet eine Nachricht an die Anwendung.

Der ActiveSync-Provider benutzt den Parameter **wParam**. Ein **wParam**-Wert von 1 zeigt an, dass der MobiLink-Provider für ActiveSync die Anwendung gestartet hat. Die Anwendung muss sich dann selbst beenden, nachdem sie die Synchronisation abgeschlossen hat. Falls die Anwendung bereits lief, als sie durch den MobiLink-Provider für ActiveSync aufgerufen wurde, ist **wParam** gleich 0. Die Anwendung kann den Parameter **wParam** ignorieren, wenn sie weiter laufen will.

Hinweise dazu, auf welchen Plattformen der Provider unterstützt wird, finden Sie unter SQL Anywhere-Komponenten nach Plattform.

Das Hinzufügen der Synchronisation hängt davon ab, ob Sie die Windows API direkt ansprechen oder die Microsoft Foundation Classes benutzen. Beide Entwicklungsmodelle werden hier beschrieben.

Siehe auch

- „Deployment des ActiveSync-Providers für UltraLite“ [*UltraLite - Datenbankverwaltung*]

ActiveSync-Synchronisation einrichten (Windows-API)

Wenn Sie direkt in der Windows API programmieren, müssen Sie die Nachricht vom MobiLink-Provider in der **WindowProc**-Funktion verarbeiten und mit der **ULIsSynchronizeMessage**-Funktion ermitteln, ob die Nachricht empfangen wurde,

Das folgende Beispiel zeigt, wie die Nachricht verarbeitet wird:

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

Dabei ist **DoSync** die Methode, die ULSynchronize aufruft.

Siehe auch

- ULIsSynchronizeMessage-Methode [UltraLite Embedded SQL] auf Seite 250

ActiveSync-Synchronisation in der main dialog-Klasse hinzufügen

Synchronisationsnachrichten in der main dialog-Klasse abfangen

Voraussetzungen

Sie müssen die Microsoft Foundation Classes benutzen, um Ihre Anwendung zu entwickeln.

Aufgabe

Ihre Anwendung muss zur Benachrichtigung einen benutzerdefinierten Windows-Klassennamen erstellen und registrieren.

1. Fügen Sie eine registrierte Nachricht hinzu und deklarieren Sie einen Message-Handler.

   Suchen Sie die Nachrichtenzuordnung in der Quelldatei Ihres Hauptdialogfeldes (der Name hat das gleiche Format wie *CMyAppDlg.cpp*). Fügen Sie eine registrierte Nachricht hinzu, indem Sie **static** verwenden, und deklarieren Sie einen Message-Handler, indem Sie ON_REGISTERED_MESSAGE verwenden, wie im folgenden Beispiel:

   ```
   static UINT WM_ULTRALITE_SYNC_MESSAGE =
    ::RegisterWindowMessage( UL_AS_SYNCHRONIZE );
   BEGIN_MESSAGE_MAP(CMyAppDlg, CDialog)
     //{{AFX_MSG_MAP(CMyAppDlg)
     //}}AFX_MSG_MAP
    ON_REGISTERED_MESSAGE( WM_ULTRALITE_SYNC_MESSAGE,
     OnDoUltraLiteSync )
   END_MESSAGE_MAP()
   ```

2. Implementieren Sie den Message-Handler.

   Fügen Sie der Hauptdialogfeldklasse mit der folgenden Signatur eine Methode hinzu. Diese Methode wird automatisch immer dann ausgeführt, wenn der MobiLink-Provider für ActiveSync eine Synchronisation von Ihrer Anwendung anfordert. Die Methode sollte die ULSynchronize-Methode aufrufen.

   ```
   LRESULT CMyAppDlg::OnDoUltraLiteSync(
      WPARAM wParam,
      LPARAM lParam
   );
   ```

   Der Rückgabewert dieser Funktion muss 0 sein.

Ergebnisse

Die main dialog-Klasse führt eine Synchronisation durch.

Siehe auch

- „Klassennamen für Anwendungen zuordnen“ auf Seite 69
- ULIsSynchronizeMessage-Methode [UltraLite Embedded SQL] auf Seite 250

ActiveSync-Synchronisation in der application-Klasse hinzufügen

Synchronisationsnachrichten in der application-Klasse abfangen.

Voraussetzungen

Sie müssen die Microsoft Foundation Classes benutzen, um Ihre Anwendung zu entwickeln.

Ihre Anwendung muss zur Benachrichtigung einen benutzerdefinierten Windows-Klassennamen erstellen und registrieren.

Aufgabe

1. Öffnen Sie den **Klassenassistenten** für die Anwendungsklasse.
2. Markieren Sie in der Liste **Nachrichten** den Eintrag **PreTranslateMessage** und klicken Sie anschließend auf **Funktion hinzufügen**.
3. Klicken Sie auf **Code bearbeiten**.

   Die PreTranslateMessage-Methode erscheint.
4. Ändern Sie die PreTranslateMessage-Methode, damit sie folgendermaßen aussieht:

```
BOOL CMyApp::PreTranslateMessage(MSG* pMsg)
{
 if( ULIsSynchronizeMessage(pMsg->message) ) {
  DoSync();
  // close application if launched by provider
  if( pMsg->wParam == 1 ) {
   ASSERT( AfxGetMainWnd() != NULL );
   AfxGetMainWnd()->SendMessage( WM_CLOSE );
  }
  return TRUE; // message has been processed
 }
 return CWinApp::PreTranslateMessage(pMsg);
}
```

   wobei die DoSync-Methode die ULSynchronize-Methode aufruft.

Ergebnisse

Die PreTranslateMessage-Methode führt eine Synchronisation durch.

Siehe auch

- „Klassennamen für Anwendungen zuordnen“ auf Seite 69
- ULSynchronize-Methode [UltraLite Embedded SQL] auf Seite 259
- ULIsSynchronizeMessage-Methode [UltraLite Embedded SQL] auf Seite 250

TCP/IP-, HTTP-, oder HTTPS-Synchronisation unter Windows Mobile

Für TCP/IP-, HTTP- oder HTTPS-Synchronisation steuert die Anwendung, wann die Synchronisation durchgeführt wird. Ihre Anwendung sollte einen Menübefehl oder ein Steuerelement auf der Benutzeroberfläche aufweisen, mit dem der Benutzer die Synchronisation anfordern kann.

Praktische Einführung: Erstellen einer Windows-Anwendung mit der C++-API

Diese praktische Einführung beschreibt, wie Sie eine UltraLite C++-Anwendung einrichten. Die Anwendung wird für Windows-PC-Betriebssysteme erstellt und von der Eingabeaufforderung ausgeführt.

Die praktische Einführung basiert auf der Entwicklung mit Microsoft Visual C++ . Sie können jedoch jede C++-Entwicklungsumgebung verwenden.

Die praktische Einführung dauert rund 30 Minuten, wenn Sie den Code kopieren und einfügen. Der letzte Abschnitt dieser praktischen Einführung enthält den vollständigen Quellcode des Programms, das hier beschrieben wird.

Kenntnisse und Erfahrungen

Für diese praktische Einführung gelten folgende Voraussetzungen:

- Sie sind mit der Programmiersprache C++ vertraut.
- Sie haben einen C++-Compiler auf Ihrem PC installiert.
- Sie können eine UltraLite-Datenbank mit dem Assistenten **Datenbank erstellen** erstellen.

Das Ziel dieser praktischen Einführung ist es, Kenntnisse über die Entwicklung einer UltraLite C++-Anwendung zu gewinnen.

Siehe auch

- „Erstellen einer UltraLite-Datenbank mit dem Assistenten Datenbank erstellen“ [*UltraLite - Datenbankverwaltung*]

Lektion 1: Erstellen und Verbinden mit einer Datenbank

In dieser Lektion erstellen Sie eine UltraLite-Datenbank. Sie schreiben und kompilieren eine C++-Anwendung und führen sie aus. Diese Anwendung greift auf die von Ihnen erstellte Datenbank zu.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie die erforderliche Software installiert haben. Siehe „Praktische Einführung: Erstellen einer Windows-Anwendung mit der C++-API“ auf Seite 75.

Aufgabe

1. Setzen Sie die **VCINSTALLDIR**-Umgebungsvariable auf das Stammverzeichnis der Visual C++-Installation, wenn die Variable noch nicht vorhanden sein sollte.

2. Fügen Sie *%VCINSTALLDIR%\VC\atlmfc\src\atl* Ihrer **INCLUDE**-Umgebungsvariablen hinzu.

3. Erstellen Sie ein Verzeichnis, das die Dateien enthalten soll, mit denen Sie während dieser praktischen Einführung arbeiten.

   In dieser praktischen Einführung wird vorausgesetzt, dass dieses Verzeichnis *c:\tutorial\cpp* lautet. Wenn Sie ein Verzeichnis mit einem anderen Namen erstellen, müssen Sie dieses Verzeichnis anstelle von *c:\tutorial\cpp* verwenden.

4. Erstellen Sie mit UltraLite in Sybase Central in Ihrem neuen Verzeichnis eine Datenbank mit dem Namen *ULCustomer.udb* und den Standardmerkmalen.

5. Fügen Sie eine Tabelle mit dem Namen **ULCustomer** zur Datenbank hinzu. Verwenden Sie die folgenden Spezifikationen für die Tabelle ULCustomer:

Spaltenname	Datentyp (Größe)	Spalte lässt NULL zu	Standardwert	Primärschlüssel
cust_id	integer	Nein	autoincrement	aufsteigend
cust_name	varchar(30)	Nein	Keine	

6. Trennen Sie die Verbindung mit der Datenbank in Sybase Central, da sich sonst Ihr Programm nicht mit der Datenbank verbinden kann.

7. Klicken Sie in Microsoft Visual C++ auf **Datei** » **Neu**.

8. Klicken Sie auf der Registerkarte **Dateien** auf **C++-Quellcodedatei**.

9. Speichern Sie die Datei als *customer.cpp* in Ihrem Übungsverzeichnis.

10. Beziehen Sie die UltraLite-Bibliotheken ein.

    Kopieren Sie den nachstehenden Code in *customer.cpp*:

```
#include <tchar.h>
#include <stdio.h>

#include "ulcpp.h"

#define MAX_NAME_LEN 100
#define MAX_ERROR_LEN 256
```

11. Definieren Sie Verbindungsparameter für die Verbindung zur Datenbank.

    In diesem Codefragment sind die Verbindungsparameter festkodiert. In einer echten Anwendung werden diese Speicherorte gegebenenfalls zur Laufzeit definiert.

    Kopieren Sie den nachstehenden Code in *customer.cpp*.

```
static ul_char const * ConnectionParms =
    "UID=DBA;PWD=sql;DBF=C:\\tutorial\\cpp\\ULCustomer.udb";
```

> **Hinweis**
> Einem Backslash im Namen der Zeichenfolge des Dateispeicherorts muss ein Backslash als Escapezeichen vorangestellt werden.

12. Legen Sie eine Methode zur Behandlung von Datenbankfehlern in der Anwendung fest.

    UltraLite bietet einen Callback-Mechanismus, um die Anwendung über Fehler zu informieren. In einer Entwicklungsumgebung kann diese Methode als Mechanismus für die Behandlung nicht erwarteter Fehler nützlich sein. Eine Produktionsanwendung umfasst normalerweise Code, der alle häufig vorkommenden Fehlersituationen verarbeiten kann. Eine Anwendung kann nach jedem Aufruf einer UltraLite-Methode eine Überprüfung auf Fehler durchführen oder eine Fehler-Callback-Funktion verwenden.

    Nachstehend wird eine Beispiel-Callback-Funktion gezeigt:

    ```
    ul_error_action UL_CALLBACK_FN MyErrorCallBack(
        const ULError *  error,
        ul_void *        user_data )
    {
        ul_error_action rc;
        an_sql_code code = error->GetSQLCode();

        (void) user_data;

        switch( code ){
            // The following error is used for flow control - don't report it
    here
            case SQLE_NOTFOUND:
                rc = UL_ERROR_ACTION_CONTINUE;
            break;

            default:
                if (code >= 0) { // warning or success
                    rc = UL_ERROR_ACTION_DEFAULT;
                } else { // negative is real error
                    ul_char etext[ MAX_ERROR_LEN ];
                    error->GetString( etext, MAX_ERROR_LEN );
                    _tprintf( "Error %ld: %s\n", code, etext );
                    rc = UL_ERROR_ACTION_CANCEL;
                }
            break;
         }
        return rc;
     }
    ```

    In UltraLite wird der Fehler SQLE_NOTFOUND häufig benutzt, um den Verarbeitungsfluss der Anwendung zu steuern. Der Fehler markiert das Ende einer Schleife in einer Ergebnismenge. Der oben dargestellte kodierte allgemeine Fehler-Handler gibt keine Fehlermeldung für diese Fehlerbedingung aus.

13. Definieren Sie eine Methode zum Öffnen einer Verbindung mit einer Datenbank.

    Wenn die Datenbankdatei nicht vorhanden ist, wird eine Fehlermeldung angezeigt. Andernfalls wird eine Verbindung hergestellt.

    ```
    static ULConnection * open_conn( void ) {
        ULConnection * conn =
    ```

```
ULDatabaseManager::OpenConnection( ConnectionParms );
    if( conn == UL_NULL ) {
        _tprintf("Unable to open existing database.\n");
    }
    return conn;
}
```

14. Implementieren Sie die Hauptmethode, die die folgenden Aufgaben ausführt:

    - Sie registriert die Fehlerbehandlungsmethode.
    - Sie öffnet eine Verbindung zu einer Datenbank.
    - Sie schließt die Verbindung und beendet den Datenbankmanager.

    ```
    int main() {
        ULConnection *    conn;

        ULDatabaseManager::Init();
        ULDatabaseManager::SetErrorCallback( MyErrorCallBack, NULL );

        conn = open_conn();
        if ( conn == UL_NULL ) {
            ULDatabaseManager::Fini();
            return 1;
        }

        // Main processing code goes here ...
        do_insert( conn );
        do_select( conn );
        do_sync( conn );

        conn->Close();
        ULDatabaseManager::Fini();
        return 0;
    }
    ```

15. Kompilieren und verknüpfen Sie die Quelldatei.

    Die Methode, die Sie benutzen, um die Quelldatei zu kompilieren, hängt von Ihrem Compiler ab. Die nachfolgenden Anweisungen gelten für den Microsoft Visual C++-Befehlszeilen-Compiler mit einer makefile-Datei:

    a. Öffnen Sie eine Eingabeaufforderung und wechseln Sie zum Verzeichnis der praktischen Einführung.

    b. Erstellen Sie eine makefile-Datei mit dem Namen *makefile*.

    c. Fügen Sie in der makefile-Datei Verzeichnisse in Ihren include-Pfad ein.

    ```
    IncludeFolders=/I"$(SQLANY16)\SDK\Include"
    ```

    d. Fügen Sie in der makefile-Datei Verzeichnisse in Ihren Bibliothekspfad ein.

    ```
    LibraryFolders=/LIBPATH:"$(SQLANY16)\UltraLite\Windows\x86\Lib\vs8"
    ```

    e. Fügen Sie in der makefile-Datei Bibliotheken zu Ihren Linker-Optionen hinzu.

    ```
    Libraries=ulimp.lib
    ```

    Die UltraLite-Laufzeitbibliothek heißt *ulimp.lib*.

f. Legen Sie in der makefile-Datei Compiler-Optionen fest. Die Optionen müssen in einer Zeile eingegeben werden.

```
CompileOptions=/c /nologo /W3 /Od /Zi /DWIN32 /DUL_USE_DLL
```

g. In der makefile-Datei fügen Sie eine Anweisung für das Linken der Anwendung hinzu:

```
customer.exe: customer.obj
   link /NOLOGO /DEBUG customer.obj $(LibraryFolders) $(Libraries)
```

h. In der makefile-Datei fügen Sie eine Anweisung für die Kompilierung der Anwendung hinzu.

```
customer.obj: customer.cpp
   cl $(CompileOptions) $(IncludeFolders) customer.cpp
```

i. Führen Sie *vsvars32.bat* aus.

```
%VCINSTALLDIR%\Tools\vsvars32.bat
```

j. Führen Sie die makefile-Datei aus.

```
nmake
```

Dadurch wird eine Programmdatei mit dem Namen *customer.exe* erstellt.

16. Führen Sie die Anwendung aus.

Geben Sie an einer Eingabeaufforderung **customer** ein.

Ergebnisse

Die Anwendung stellt eine Verbindung mit der Datenbank her und trennt diese anschließend. Die Anwendung wird erfolgreich ausgeführt, wenn keine Fehlermeldungen angezeigt werden.

Nächste Schritte

Gehen Sie weiter zu „Lektion 2: Einfügen von Daten in die Datenbank“ auf Seite 79.

Siehe auch

- „Erstellen einer UltraLite-Datenbank mit dem Assistenten Datenbank erstellen“ [*UltraLite - Datenbankverwaltung*]
- „UltraLite-Verbindungsparameter“ [*UltraLite - Datenbankverwaltung*]
- „Fehlerbehandlung“ auf Seite 22

Lektion 2: Einfügen von Daten in die Datenbank

In dieser Lektion fügen Sie einer Datenbank Daten hinzu.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „Lektion 1: Erstellen und Verbinden mit einer Datenbank“ auf Seite 75.

Aufgabe

1. Fügen Sie die nachstehende Methode in *customer.cpp* unmittelbar vor der Hauptmethode ein:

```
static bool do_insert( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        _tprintf( "Table not found: ULCustomer\n" );
        return false;
    }
    if( table->GetRowCount() == 0 ) {
        _tprintf( "Inserting one row.\n" );
        table->InsertBegin();
        table->SetString( "cust_name", "New Customer" );
        table->Insert();
        conn->Commit();
    } else {
        _tprintf( "The table has %lu rows\n", table->GetRowCount() );
    }
    table->Close();
    return true;
}
```

   Diese Methode führt die folgenden Aufgaben aus.

   - Sie öffnet die Tabelle mit der connection->OpenTable()-Methode. Sie müssen ein Table-Objekt öffnen, um Vorgänge in der Tabelle ausführen zu können.
   - Wenn die Tabelle leer ist, fügt sie ihr eine Zeile hinzu. Um eine Zeile einzufügen, ändert der Code den Modus mit InsertBegin auf den Einfügemodus, legt Werte für jede erforderliche Spalte fest und führt eine Einfügung durch, um die Zeile der Datenbank hinzuzufügen.
   - Wenn die Tabelle nicht leer ist, meldet sie die Anzahl der Zeilen in der Tabelle.
   - Sie schließt das Table-Objekt, um ihm zugeordnete Ressourcen freizugeben.
   - Sie gibt einen booleschen Wert zurück, der anzeigt, ob der Vorgang erfolgreich war.

2. Rufen Sie die von Ihnen erstellte do_insert-Methode auf.

   Fügen Sie die folgende Zeile unmittelbar nach dem Aufruf von conn->Close der `main()`-Methode hinzu.

```
do_insert(conn);
```

3. Kompilieren Sie Ihre Anwendung mit "nmake".

4. Führen Sie Ihre Anwendung aus, indem Sie an einer Eingabeaufforderung **customer** eingeben.

Ergebnisse

Die Anwendung wird ausgeführt und Sie können Daten in die ULCustomer-Tabelle einfügen.

Nächste Schritte

Gehen Sie weiter zu „Lektion 3: Auswählen und Auflisten von Zeilen aus der Tabelle“ auf Seite 81.

Lektion 3: Auswählen und Auflisten von Zeilen aus der Tabelle

In dieser Lektion rufen Sie Zeilen aus der Tabelle ab und geben sie auf der Befehlszeile aus.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „Lektion 1: Erstellen und Verbinden mit einer Datenbank“ auf Seite 75.

Aufgabe

1. Fügen Sie nachstehende Methode in *customer*. *cpp* unmittelbar nach der do_insert-Methode ein. Diese Methode führt die folgenden Aufgaben aus:

   - Sie öffnet die Tabelle.
   - Sie ruft die Spaltenbezeichner ab.
   - Sie setzt die aktuelle Position vor die erste Zeile der Tabelle.

     Alle Vorgänge in der Tabelle werden an der aktuellen Position durchgeführt. Die Position kann vor der ersten Zeile, in einer der Zeilen oder hinter der letzten Zeile sein. Standardmäßig, wie auch in diesem Fall, sind die Zeilen anhand ihrer Primärschlüsselwerte (cust_id) geordnet. Um die Zeilen anders anzuordnen, können Sie einer UltraLite-Datenbank einen Index hinzufügen und eine Tabelle unter Verwendung dieses Indexes öffnen.
   - Für jede Zeile werden die Werte für cust_ID und cust_name ausgegeben. Die Schleife läuft, bis die Next-Methode FALSE zurückgibt, was nach der letzten Zeile eintritt.
   - Sie schließt das Table-Objekt.

   ```
   static bool do_select( ULConnection * conn )
   {
       ULTable * table = conn->OpenTable( "ULCustomer" );
       if( table == UL_NULL ) {
           return false;
       }
       ULTableSchema * schema = table->GetTableSchema();
       if( schema == UL_NULL ) {
           table->Close();
           return false;
       }
       ul_column_num id_cid =
           schema->GetColumnID( "cust_id" );
       ul_column_num cname_cid =
           schema->GetColumnID( "cust_name" );
       schema->Close();

       _tprintf( "\n\nTable 'ULCustomer' row contents:\n" );
       while( table->Next() ) {
           ul_char cname[ MAX_NAME_LEN ];
           table->GetString( cname_cid, cname, MAX_NAME_LEN );
           _tprintf( "id=%d, name=%s \n", (int)table->GetInt(id_cid),
   cname );
       }
       table->Close();
   ```

```
        return true;
    }
```

2. Fügen Sie die folgende Zeile unmittelbar hinter dem Aufruf der insert-Methode zur main-Methode hinzu:

   ```
   do_select(conn);
   ```

3. Kompilieren Sie Ihre Anwendung mit *nmake*.

4. Führen Sie Ihre Anwendung aus, indem Sie an einer Eingabeaufforderung *customer* eingeben.

Ergebnisse

Eine Liste aller Kunden-IDs und Kundennamen in der ULCustomer-Tabelle wird ausgegeben.

Nächste Schritte

Gehen Sie weiter zu „Lektion 4: Hinzufügen von Synchronisation zu Ihrer Anwendung“ auf Seite 82.

Lektion 4: Hinzufügen von Synchronisation zu Ihrer Anwendung

In dieser Lektion fügen Sie Ihrer Anwendung Synchronisationscode hinzu, starten den MobiLink-Server und führen Ihre Anwendung aus, um die Synchronisation mit der konsolidierten Datenbank durchzuführen.

Voraussetzungen

In dieser Lektion wird davon ausgegangen, dass Sie bereits alle vorherigen Lektionen abgeschlossen haben. Siehe „Lektion 1: Erstellen und Verbinden mit einer Datenbank“ auf Seite 75.

Kontext und Bemerkungen

Die in den vorhergehenden Lektionen von Ihnen erstellte UltraLite-Datenbank wird mit der UltraLite 16-Beispieldatenbank synchronisiert. Die UltraLite 16-Beispieldatenbank verfügt über die Tabelle ULCustomer, deren Spalten diejenigen aus der customer-Tabelle Ihrer lokalen UltraLite-Datenbank umfassen.

Aufgabe

1. Fügen Sie nachstehende Methode der Datei *customer.cpp* hinzu. Diese Methode führt die folgenden Aufgaben aus:

   - Aktiviert die TCP/IP-Kommunikation durch Aufruf von EnableTcpipSynchronization. Die Synchronisation kann auch über HTTP, HTTPS und TLS durchgeführt werden.
   - Sie definiert die Skriptversion. Die MobiLink-Synchronisation wird von Skripten in der konsolidierten Datenbank gesteuert. Die Skriptversion legt fest, welche Gruppe von Skripten verwendet werden soll.

- Sie legt den MobiLink-Benutzernamen fest. Dieser Wert wird zur Authentifizierung auf dem MobiLink-Server verwendet. Sie unterscheidet sich von der Benutzer-ID der UltraLite-Datenbank. Sie können aber bei Ihren Anwendungen denselben Wert für die Benutzer-ID verwenden.
- Setzt den download_only-Parameter auf TRUE. Standardmäßig ist die MobiLink-Synchronisation bidirektional. Diese Anwendung verwendet die reine Downloadsynchronisation, damit die Zeilen in Ihrer Tabelle nicht in die Beispieldatenbank geladen werden.

```
static bool do_sync( ULConnection * conn )
{
    ul_sync_info info;
    ul_stream_error * se = &info.stream_error;

    ULDatabaseManager::EnableTcpipSynchronization();
    conn->InitSyncInfo( &info );
    info.stream = "TCPIP";
    info.version = "custdb 12.0";
    info.user_name = "50";
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( "Synchronization error \n" );
        _tprintf( "  stream_error_code is '%lu'\n", se-
>stream_error_code );
        _tprintf( "  system_error_code is '%ld'\n", se-
>system_error_code );
        _tprintf( "  error_string is '" );
        _tprintf( "%s", se->error_string );
        _tprintf( "'\n" );
        return false;
    }
    return true;
}
```

2. Fügen Sie der main-Methode nach dem do_select-Methodenaufruf folgende Codezeile hinzu:

   ```
   do_sync(conn);
   ```

3. Kompilieren Sie Ihre Anwendung mit *nmake*.

4. Starten Sie den MobiLink-Server.

   Führen Sie an der Eingabeaufforderung folgenden Befehl aus:

   ```
   mlsrv16 -c "dsn=SQL Anywhere 16 CustDB;uid=ml_server;pwd=sql" -v -vr -vs -
   zu+ -o custdbASA.log
   ```

   Die Option -zu+ ermöglicht das automatische Hinzufügen von Benutzern. Die Option -v+ aktiviert die ausführliche Protokollierung aller Meldungen.

5. Führen Sie Ihre Anwendung aus, indem Sie an einer Eingabeaufforderung *customer* eingeben.

Ergebnisse

Im Fenster des MobiLink-Servers werden Statusmeldungen angezeigt, die den Synchronisationsfortschritt beschreiben. Wenn die Synchronisation erfolgreich verläuft, zeigt die letzte Meldung `Synchronisation abgeschlossen` an.

Siehe auch

- „UltraLite-Clients“ [*UltraLite - Datenbankverwaltung*]
- „MobiLink-Serveroptionen“ [*MobiLink - Serveradministration*]

Programmcode der praktischen Einführung

Im Folgenden ist der vollständige Code des Programms aus der praktischen Einführung, das in den vorangehenden Abschnitten beschrieben wurde, aufgelistet.

```
#include <tchar.h>
#include <stdio.h>

#include "ulcpp.h"

#define MAX_NAME_LEN 100
#define MAX_ERROR_LEN 256

static ul_char const * ConnectionParms =
    "UID=DBA;PWD=sql;DBF=c:\\tutorial\\cpp\\ULCustomer.udb";

ul_error_action UL_CALLBACK_FN MyErrorCallBack(
    const ULError *  error,
    ul_void *        user_data )
{
    ul_error_action rc;
    an_sql_code code = error->GetSQLCode();

    (void) user_data;

    switch( code ){
        // The following error is used for flow control - don't report it
here
        case SQLE_NOTFOUND:
            rc = UL_ERROR_ACTION_CONTINUE;
        break;

        default:
            if (code >= 0) { // warning or success
                rc = UL_ERROR_ACTION_DEFAULT;
            } else { // negative is real error
                ul_char etext[ MAX_ERROR_LEN ];
                error->GetString( etext, MAX_ERROR_LEN );
                _tprintf( "Error %ld: %s\n", code, etext );
                rc = UL_ERROR_ACTION_CANCEL;
            }
        break;
     }
    return rc;
}

static ULConnection * open_conn( void ) {
    ULConnection * conn =
ULDatabaseManager::OpenConnection( ConnectionParms );
    if( conn == UL_NULL ) {
        _tprintf( "Unable to open existing database.\n" );
    }
    return conn;
}
```

```
static bool do_insert( ULConnection * conn ) {
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        _tprintf( "Table not found: ULCustomer\n" );
        return false;
    }
    if( table->GetRowCount() == 0 ) {
        _tprintf( "Inserting one row.\n" );
        table->InsertBegin();
        table->SetString( "cust_name", "New Customer" );
        table->Insert();
        conn->Commit();
    } else {
        _tprintf( "The table has %lu rows\n",
        table->GetRowCount() );
    }
    table->Close();
    return true;
}

static bool do_select( ULConnection * conn )
{
    ULTable * table = conn->OpenTable( "ULCustomer" );
    if( table == UL_NULL ) {
        return false;
    }
    ULTableSchema * schema = table->GetTableSchema();
    if( schema == UL_NULL ) {
        table->Close();
        return false;
    }
    ul_column_num id_cid =
        schema->GetColumnID( "cust_id" );
    ul_column_num cname_cid =
        schema->GetColumnID( "cust_name" );

    schema->Close();

    _tprintf( "\n\nTable 'ULCustomer' row contents:\n" );

    while( table->Next() ) {
        ul_char cname[ MAX_NAME_LEN ];

        table->GetString( cname_cid, cname, MAX_NAME_LEN );

        _tprintf( "id=%d, name=%s \n", (int)table->GetInt(id_cid), cname );
    }
    table->Close();
    return true;
}

static bool do_sync( ULConnection * conn )
{
    ul_sync_info info;
    ul_stream_error * se = &info.stream_error;

    ULDatabaseManager::EnableTcpipSynchronization();
    conn->InitSyncInfo( &info );
    info.stream = "TCPIP";
    info.version = "custdb 12.0";
    info.user_name = "50";
    info.download_only = true;
    if( !conn->Synchronize( &info ) ) {
        _tprintf( "Synchronization error \n" );
```

```
            _tprintf( "  stream_error_code is '%lu'\n", se->stream_error_code );
            _tprintf( "  system_error_code is '%ld'\n", se->system_error_code );
            _tprintf( "  error_string is '" );
            _tprintf( "%s", se->error_string );
            _tprintf( "'\n" );
            return false;
        }
        return true;
    }

    int main()
    {
        ULConnection *    conn;

        ULDatabaseManager::Init();
        ULDatabaseManager::SetErrorCallback( MyErrorCallBack, NULL );

        conn = open_conn();
        if( conn == UL_NULL ){
            ULDatabaseManager::Fini();
            return 1;
        }

        // Main processing code goes here ...
        do_insert( conn );
        do_select( conn );
        do_sync( conn );

        conn->Close();
        ULDatabaseManager::Fini();
        return 0;
    }
```

API-Referenz

In diesem Abschnitt finden Sie die UltraLite für C/C++-API

UltraLite C/C++ - Gemeinsame API-Referenz

Dieser Abschnitt listet die Funktionen und Makros auf, die Sie mit Embedded SQL oder mit der C++-Schnittstelle verwenden können. Für die meisten Funktionen in diesem Abschnitt ist ein SQL-Kommunikationsbereich (SQLCA) erforderlich.

Header-Datei

- `ulglobal.h`

Makros und Compiler-Direktiven für UltraLite-C/C++-Anwendungen

Sofern nicht anders angegeben, beziehen sich die Direktiven sowohl auf Embedded SQL- als auch auf C++-API-Anwendungen.

Sie können Compilerdirektiven wie folgt angeben:

- In Ihrer Compiler-Befehlszeile. Eine Direktive wird gewöhnlich mit der Option /D angegeben. Um z.B. eine UltraLite-Anwendung mit Benutzerauthentifizierung zu kompilieren, sieht ein Makefile für den Microsoft Visual C++-Compiler möglicherweise wie folgt aus:

  ```
  CompileOptions=/c /DPRWIN32 /Od /Zi /DWIN32
  /DUL_USE_DLL

  IncludeFolders= \
  /I"$(VCDIR)\include" \
  /I"$(SQLANY16)\SDK\Include"

  sample.obj: sample.cpp
   cl $(CompileOptions) $(IncludeFolders) sample.cpp
  ```

  Dabei gilt: VCDIR *ist das Visual C++-Verzeichnis und* SQLANY16 ist das SQL Anywhere-Installationsverzeichnis.

- Im Fenster der Compiler-Einstellungen auf der Benutzeroberfläche.

- Im Quellcode. Direktiven werden mit der Anweisung `#define` angegeben.

UL_USE_DLL-Makro

Legt fest, dass die Anwendung die Laufzeitbibliotheks-DLL verwendet, anstelle einer statischen Laufzeitbibliothek

Bemerkungen

Ist für Windows Mobile- und Windows-Anwendungen verfügbar.

UNDER_CE-Makro

Standardmäßig ist dieser Makro in allen neuen Visual C++-Projekten für Smart-Devices definiert.

Bemerkungen

Ist für Windows Mobile-Anwendungen verfügbar.

Siehe auch

- „UltraLite-Anwendungsentwicklung für Windows Mobile“ auf Seite 67

Beispiel

```
/D UNDER_CE
```

UL_RS_STATE-Enumeration

Gibt möglichen Ergebnismengen- oder Cursorstatus zurück.

Syntax

```
public enum UL_RS_STATE
```

Mitglieder

Mitgliedsname	Beschreibung
UL_RS_STATE_ERROR	Fehler.
UL_RS_STATE_UNPREPARED	Nicht vorbereitet.
UL_RS_STATE_ON_ROW	In einer gültigen Zeile.
UL_RS_STATE_BEFORE_FIRST	Vor der ersten Zeile.
UL_RS_STATE_AFTER_LAST	Hinter der letzten Zeile.
UL_RS_STATE_COMPLETED	Geschlossen.

ul_column_sql_type-Enumeration

Repräsentiert die SQL-Datentypen für eine Spalte.

Syntax

```
public enum ul_column_sql_type
```

Mitglieder

Mitgliedsname	Beschreibung
UL_SQLTYPE_BAD_INDEX	Gibt an, dass die Spalte beim angegebenen Index nicht vorhanden ist.
UL_SQLTYPE_S_LONG	Gibt an, dass die Spalte eine Langzahl mit Vorzeichen enthält.
UL_SQLTYPE_U_LONG	Gibt an, dass die Spalte eine Langzahl ohne Vorzeichen enthält.
UL_SQLTYPE_S_SHORT	Gibt an, dass die Spalte eine Kurzzahl mit Vorzeichen enthält.
UL_SQLTYPE_U_SHORT	Gibt an, dass die Spalte eine Kurzzahl ohne Vorzeichen enthält.
UL_SQLTYPE_S_BIG	Gibt an, dass die Spalte eine 64-Bit-Ganzzahl mit Vorzeichen enthält.
UL_SQLTYPE_U_BIG	Gibt an, dass die Spalte eine 64-Bit-Ganzzahl ohne Vorzeichen enthält.
UL_SQLTYPE_TINY	Gibt an, dass die Spalte eine 8-Bit-Ganzzahl ohne Vorzeichen enthält.
UL_SQLTYPE_BIT	Gibt an, dass die Spalte einen 1-Bit-Parameter enthält.
UL_SQLTYPE_TIMESTAMP	Gibt an, dass die Spalte Zeitstempelinformationen enthält.
UL_SQLTYPE_DATE	Gibt an, dass die Spalte Datumsinformationen enthält.
UL_SQLTYPE_TIME	Gibt an, dass die Spalte Zeitinformationen enthält.
UL_SQLTYPE_DOUBLE	Gibt an, dass die Spalte eine doppeltgenaue Gleitkommazahl (8 Byte) enthält.
UL_SQLTYPE_REAL	Gibt an, dass die Spalte eine einfachgenaue Gleitkommazahl (4 Byte) enthält.
UL_SQLTYPE_NUMERIC	Gibt an, dass die Spalte exakte nummerische Daten enthält, mit festgelegter Gesamtstellen- und Dezimalstellenzahl.
UL_SQLTYPE_BINARY	Gibt an, dass die Spalte Binärdaten mit festgelegter maximalen Länge enthält.
UL_SQLTYPE_CHAR	Gibt an, dass die Spalte Zeichendaten mit festgelegter Länge enthält.
UL_SQLTYPE_LONGVARCHAR	Gibt an, dass die Spalte Zeichendaten mit variabler Länge enthält.

Mitgliedsname	Beschreibung
UL_SQLTYPE_LONGBINARY	Gibt an, dass die Spalte Binärdaten mit variabler Länge enthält.
UL_SQLTYPE_UUID	Gibt an, dass die Spalte eine UUID enthält.
UL_SQLTYPE_ST_GEOMETRY	Gibt an, dass die Spalte räumliche Daten in Form von Punkten enthält.
UL_SQLTYPE_TIME-STAMP_WITH_TIME_ZONE	Gibt an, dass die Spalte Zeitstempel- und Zeitzoneninformationen enthält.

Bemerkungen

Diese Werte entsprechen den SQL-Spaltentypen.

ul_column_storage_type-Enumeration

Repräsentiert die Hostvariablentypen für eine Spalte.

Syntax

```
public enum ul_column_storage_type
```

Mitglieder

Mitgliedsname	Beschreibung
UL_TYPE_BAD_INDEX	Gibt einen ungültigen Wert an.
UL_TYPE_S_LONG	Gibt einen ul_s_long-Wert an (32-Bit-Ganzzahl mit Vorzeichen).
UL_TYPE_U_LONG	Gibt einen ul_u_long-Wert an (32-Bit-Ganzzahl ohne Vorzeichen).
UL_TYPE_S_SHORT	Gibt einen ul_s_short-Wert an (16-Bit-Ganzzahl mit Vorzeichen).
UL_TYPE_U_SHORT	Gibt einen ul_u_short-Wert an (16-Bit-Ganzzahl ohne Vorzeichen).
UL_TYPE_S_BIG	Gibt einen ul_s_big-Wert an (64-Bit-Ganzzahl mit Vorzeichen).
UL_TYPE_U_BIG	Gibt einen ul_u_big-Wert an (64-Bit-Ganzzahl ohne Vorzeichen).
UL_TYPE_TINY	Gibt einen ul_byte-Wert an (8 Bit, ohne Vorzeichen).
UL_TYPE_BIT	Gibt einen ul_byte-Wert an (8 Bit ohne Vorzeichen, 1 Bit verwendet).
UL_TYPE_DOUBLE	Gibt einen ul_double-Wert an (double).

Mitgliedsname	Beschreibung
UL_TYPE_REAL	Gibt einen ul_real-Wert an (float).
UL_TYPE_BINARY	Gibt einen ul_binary-Wert an (2 Byte Länge, gefolgt von Byte-Array).
UL_TYPE_TIME-STAMP_STRUCT	Gibt einen DECL_DATETIME-Wert an.
UL_TYPE_TCHAR	Gibt ein Zeichen-Array an (Zeichenfolgenpuffer).
UL_TYPE_CHAR	Gibt ein Zeichen-Array an (Zeichenfolgenpuffer).
UL_TYPE_WCHAR	Gibt ein ul_wchar-Array an (UTF16).
UL_TYPE_GUID	Gibt eine GUID-Struktur an.

Bemerkungen

Diese Werte werden verwendet, um den Hostvariablentyp zu identifizieren, der für eine Spalte erforderlich ist, und anzuzeigen, wie UltraLite die Werte abrufen soll.

ul_error_action-Enumeration

Legt mögliche Fehleraktionen fest, die vom Callback zurückgegeben werden.

Syntax

```
public enum ul_error_action
```

Mitglieder

Mitgliedsname	Beschreibung
UL_ERROR_ACTION_DEFAULT	So verhalten, als ob es keinen Fehler-Callback gibt.
UL_ERROR_ACTION_CANCEL	Vorgang abbrechen, der zu dem Fehler geführt hat.
UL_ERROR_ACTION_TRY_AGAIN	Vorgang wiederholen, der zu dem Fehler geführt hat
UL_ERROR_ACTION_CONTINUE	Fortsetzen und den Vorgang ignorieren, der zu dem Fehler geführt hat.

Bemerkungen

Nicht alle Aktionen gelten für alle Fehlercodes.

ul_sync_state-Enumeration

Gibt die aktuelle Stufe einer Synchronisation an.

Syntax

```
public enum ul_sync_state
```

Mitglieder

Mitgliedsname	Beschreibung
UL_SYNC_STATE_STARTING	Die Synchronisation wird gestartet, die einleitende Parametervalidierung ist abgeschlossen und das Synchronisationsergebnis wird gespeichert.
UL_SYNC_STATE_CONNECTING	Verbindung mit dem MobiLink-Server.
UL_SYNC_STATE_SENDING_HEADER	Die Synchronisationsverbindung ist hergestellt und die ersten Daten werden demnächst gesendet.
UL_SYNC_STATE_SENDING_TABLE	Eine Tabelle wird demnächst gesendet.
UL_SYNC_STATE_SENDING_DATA	Schemainformationen oder Zeilendaten werden gesendet.
UL_SYNC_STATE_FINISHING_UPLOAD	Die Uploadphase ist abgeschlossen und Statusinformationen werden demnächst festgeschrieben.
UL_SYNC_STATE_RECEIVING_UPLOAD_ACK	Lesen der Daten vom Server beginnt mit der Uploadquittierung.
UL_SYNC_STATE_RECEIVING_TABLE	Eine Tabelle wird demnächst empfangen.
UL_SYNC_STATE_RECEIVING_DATA	Daten für die zuletzt identifizierte Tabelle werden empfangen.
UL_SYNC_STATE_COMMITTING_DOWNLOAD	Die Downloadphase ist abgeschlossen und die eingelesenen Zeilen werden festgeschrieben.
UL_SYNC_STATE_ROLLING_BACK_DOWNLOAD	Ein Fehler ist beim Download aufgetreten und der Download wird zurückgesetzt.
UL_SYNC_STATE_SENDING_DOWNLOAD_ACK	Es wird gerade eine Bestätigung gesendet, dass der Download-Vorgang abgeschlossen wurde.

Mitgliedsname	Beschreibung
UL_SYNC_STATE_DISCONNECTING	Die Verbindung mit dem Server wird gerade getrennt.
UL_SYNC_STATE_DONE	Die Synchronisation wurde erfolgreich abgeschlossen.
UL_SYNC_STATE_ERROR	Die Synchronisation wurde abgeschlossen, jedoch mit einem Fehler.

Bemerkungen

Gehen Sie nicht von der Annahme aus, dass die Synchronisationsstatusmeldungen in der angegebenen Reihenfolge auftreten.

ul_validate_status_id-Enumeration

Legt mögliche Status-IDs für das UltraLite-Validierungstool fest.

Syntax

```
public enum ul_validate_status_id
```

Mitglieder

Mitgliedsname	Beschreibung	Wert
UL_VALID_NO_ERROR	Es ist kein Fehler aufgetreten.	0
UL_VALID_START	Starten der Validierung.	1
UL_VALID_END	Ende der Validierung. Parm1 enthält den sich ergebenden SQLCODE, der Erfolg oder Fehlschlag anzeigt.	2
UL_VALID_CHECKING_PAGE	Sendet eine periodische Statusmeldung während der Prüfung von Datenbankseiten. Parm1 enthält eine mit der Seite verknüpfte Zahl. Die Reihenfolge ist nicht definiert.	10
UL_VALID_CHECKING_TABLE	Prüfung einer Tabelle. Parm1 enthält den Tabellennamen.	20

Mitgliedsname	Beschreibung	Wert
UL_VALID_CHECKING_INDEX	Prüfung eines Indexes. Parm1 speichert den Tabellennamen und parm2 speichert den Indexnamen.	21
UL_VALID_HASH_REPORT	Berichte über die Index-Hash-Nutzung. (Nur Entwicklerversion) Parm1 enthält den Tabellennamen, parm2 den Indexnamen, parm3 die Anzahl sichtbarer Zeilen, parm4 die Anzahl der eindeutigen Hashwerte und parm5 die maximale Anzahl des Auftretens eines Hash-Eintrags.	30
UL_VALID_REDUNDANT_INDEX	Ein redundanter Index wurde gefunden. (Nur Entwicklerversion) Parm1 enthält den Tabellennamen, parm2 den redundanten Indexnamen und parm3 den Namen des Indexes, der den anderen redundant macht.	31
UL_VALID_DUPLICATE_INDEX	Zwei Indizes sind identisch. (Nur Entwicklerversion) Parm1 enthält den Tabellennamen, parm2 den Namen des ersten Indexes und parm3 den Namen des zweiten Indexes.	32
UL_VALID_DATABASE_ERROR	Beim Zugriff auf die Datenbank ist ein Fehler aufgetreten. Weitere Informationen finden Sie im SQLCODE.	100
UL_VALID_STARTUP_ERROR	Fehler beim Starten der Datenbank. (bei systemnahen Zugriff)	101
UL_VALID_CONNECT_ERROR	Fehler beim Verbinden mit der Datenbank.	102
UL_VALID_INTERRUPTED	Validierungsvorgang unterbrochen.	103
UL_VALID_CORRUPT_PAGE_TABLE	Seitentabelle ist beschädigt.	110
UL_VALID_FAILED_CHECKSUM	Seitenprüfsumme ist fehlgeschlagen. Parm1 enthält eine mit der Seite verknüpfte Zahl	111
UL_VALID_CORRUPT_PAGE	Eine Seite ist beschädigt. Parm1 enthält eine mit der Seite verknüpfte Zahl.	112

Mitgliedsname	Beschreibung	Wert
UL_VALID_ROWCOUNT_MISMATCH	Die Anzahl der Zeilen im Index unterscheidet sich von der Tabellenzeilenanzahl. Parm1 enthält den Tabellennamen und parm2 den Indexnamen.	120
UL_VALID_BAD_ROWID	Im Index ist ein ungültiger Zeilenbezeichner vorhanden. Parm1 enthält den Tabellennamen und parm2 den Namen des Indexnamens.	121

ul_binary-Struktur

Setzt Binärwerte und ruft sie aus einer Tabelle in der Datenbank ab.

Syntax

```
public typedef struct ul_binary
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
data	ul_byte	Die aktuellen Daten, die (bei Insert) geschrieben werden sollen, oder (bei Select) ausgelesen wurden.
len	ul_length	Die Anzahl von Bytes im Wert.

ul_error_info-Struktur

Speichert vollständige Informationen über einen UltraLite-Fehler.

Syntax

```
public typedef struct ul_error_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
sqlcode	an_sql_code	Der SQLCODE-Wert.
sqlcount	ul_s_long	Der SQLCOUNT-Wert.

Siehe auch

- ULErrorInfoString-Methode [UltraLite Embedded SQL] auf Seite 243
- ULErrorInfoURL-Methode [UltraLite Embedded SQL] auf Seite 244
- ULErrorInfoInitFromSqlca-Methode [UltraLite Embedded SQL] auf Seite 242
- ULErrorInfoParameterCount-Methode [UltraLite Embedded SQL] auf Seite 243
- ULErrorInfoParameterAt-Methode [UltraLite Embedded SQL] auf Seite 243

ul_stream_error-Struktur

Speichert Fehlerinformationen des Kommunikationsdatenstroms während der Synchronisation.

Syntax

```
public typedef struct ul_stream_error
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
error_string	char	Eine Zeichenfolge mit zusätzlichen Informationen, sofern verfügbar, für den stream_error_code-Wert.
stream_error_code	ss_error_code	Der spezifische Datenstromfehler. Siehe ss_error_code.
system_error_code	asa_int32	Ein systemspezifischer Fehlercode. Weitere Hinweise zu Fehlercodes finden Sie in der Dokumentation zu Ihrer Plattform.

ul_sync_info-Struktur

Speichert Synchronisationsdaten.

Syntax

```
public typedef struct ul_sync_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
additional_parms	const char *	Eine Zeichenfolge von Namen-Wert-Paaren mit zusätzlichen Parametern.
auth_parms	const char *	Stellt ein Array von Authentifizierungsparametern in MobiLink-Ereignissen bereit.

Mitgliedsname	Typ	Beschreibung
auth_status	ul_auth_status	Der Status der MobiLink-Benutzerauthentifizierung. Der MobiLink-Server gibt diese Information an den Client weiter.
auth_value	ul_s_long	Die Ergebnisse eines benutzerdefinierten Benutzerauthentifizierungsskripts in MobiLink. Der MobiLink-Server stellt diese Informationen dem Client bereit, um den Authentifzierungsstatus zu ermitteln.
download_only	ul_bool	Führt während der aktuellen Synchronisation keinen Upload der Änderungen in der UltraLite-Datenbank durch.
ignored_rows	ul_bool	Der Status der ignorierten Zeilen. Dieses schreibgeschützte Feld meldet TRUE, wenn Zeilen vom MobiLink-Server während der Synchronisation aufgrund fehlender Skripten ignoriert wurden.
init_verify	ul_sync_info *	Initialisiert die Verifizierung.
keep_partial_download	ul_bool	Wenn ein Download wegen eines Kommunikationsfehlers während der Synchronisation fehlschlägt, steuert dieser Parameter, ob UltraLite den teilweisen Download fortsetzt oder die Änderungen zurücksetzt.
new_password	const char *	Eine Zeichenfolge, die ein neues MobiLink-Kennwort festlegt, das dem Benutzernamen zugeordnet ist. Dieser Parameter ist optional.
num_auth_parms	ul_byte	Die Anzahl der Authentifizierungsparameter, die an die Authentifizierungsparameter in MobiLink-Ereignissen übergeben werden.
observer	ul_sync_observer_fn	Ein Zeiger auf eine Callback-Funktion oder einen Event-Handler, der die Synchronisation überwacht. Dieser Parameter ist optional.
partial_download_retained	ul_bool	Wenn ein Download wegen eines Kommunikationsfehlers während der Synchronisation fehlschlägt, wird durch diesen Parameter angegeben, ob UltraLite die geladenen Änderungen übernommen anstatt zurückgesetzt hat.
password	const char *	Eine Zeichenfolge, die das vorhandene MobiLink-Kennwort festlegt, das mit dem Benutzernamen verbunden ist. Dieser Parameter ist optional.

Mitgliedsname	Typ	Beschreibung
ping	ul_bool	Bestätigung von Kommunikationen zwischen dem UltraLite-Client und dem MobiLink-Server. Wenn dieser Parameter auf TRUE gesetzt wird, findet keine Synchronisation statt.
publications	const char *	Eine kommagetrennte Liste von Publikationen, die anzeigt, welche Daten in die Synchronisation einzubeziehen sind.
resume_partial_download	ul_bool	Setzt einen fehlgeschlagenen Download fort. Die Synchronisation sendet keine Änderungen beim Upload und lädt nur die Änderungen, die beim fehlgeschlagenen Download geladen werden sollten.
send_download_ack	ul_bool	Informiert den MobiLink-Server, ob der Client Downloadbestätigungen sendet.
stream	const char *	Das MobiLink-Netzwerkprotokoll zur Verwendung in der Synchronisation.
stream_error	ul_stream_error	Die Struktur, die Berichtsinformationen zu Kommunikationsfehlern enthält.
stream_parms	const char *	Die Optionen zum Konfigurieren des ausgewählten Netzwerkprotokolls.
upload_ok	ul_bool	Der Status der Daten, die auf den MobiLink-Server übertragen wurden. Dieses Feld hat den Wert TRUE, wenn der Upload erfolgreich war.
upload_only	ul_bool	Lädt während der aktuellen Synchronisation keine Änderungen von der konsolidierten Datenbank herunter. Dies kann Verbindungszeit sparen, vor allem bei langsamen Verbindungen.
user_data	ul_void *	Übergabe von anwendungsspezifischen Informationen an die Synchronisations-Beobachtungsfunktion. Dieser Parameter ist optional.
user_name	const char *	Eine Zeichenfolge, die der MobiLink-Server verwendet, um einen eindeutigen MobiLink-Benutzer zu identifizieren.
version	const char *	Die Versionszeichenfolge gestattet es einer UltraLite-Anwendung, zwischen einer Reihe von Synchronisationsskripten zu wählen.

Bemerkungen

Die Synchronisationsparameter steuern das Synchronisationsverhalten zwischen einer UltraLite-Datenbank und dem MobiLink-Server. Dazu sind die Synchronisationsparameter Stream Type, User Name und Version erforderlich. Wenn Sie diese nicht setzen, gibt die Synchronisationsmethode einen Fehler zurück (SQLE_SYNC_INFO_INVALID oder einen äquivalenten Fehler). Es darf jeweils nur einer der Parameter Download Only, Ping oder Upload Only angegeben werden. Wenn Sie mehrere dieser Parameter auf TRUE setzen, gibt die Synchronisationsmethode einen Fehler zurück (SQLE_SYNC_INFO_INVALID oder einen äquivalenten Fehler).

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

ul_sync_result-Struktur

Speichert das Synchronisationsergebnis, damit die geeignete Aktion in der Anwendung durchgeführt werden kann.

Syntax

```
public typedef struct ul_sync_result
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
auth_status	ul_auth_status	Der Status der Synchronisationsauthentifizierung.
auth_value	ul_s_long	Der vom MobiLink-Server zur Ermittlung des auth_status-Ergebnisses verwendete Wert.
error_status	ul_error_info	Der Fehlerstatus der letzten Synchronisation.
ignored_rows	ul_bool	TRUE, wenn die übertragenen Zeilen ignoriert wurden, andernfalls FALSE
partial_download_retained	ul_bool	Der Wert, der anzeigt, dass ein Teil-Download gespeichert wurde. Siehe "keep_partial_download".
received	ul_sync_stats_download	Downloadstatistiken.
sent	ul_sync_stats_upload	Uploadstatistiken.
stream_error	ul_stream_error	Die Fehlerinformationen des Kommunikationsdatenstroms.
timestamp	SQLDATETIME	Datum und Zeit der letzten Synchronisation.

Mitgliedsname	Typ	Beschreibung
upload_ok	ul_bool	Wird bei erfolgreichem Upload auf TRUE gesetzt, andernfalls FALSE

ul_sync_stats_download-Struktur

Gibt die Download-Statistik des Synchronisationsdatenstroms zurück.

Syntax

```
public typedef struct ul_sync_stats_download
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
bytes	ul_u_long	Die Anzahl der aktuell empfangenen Byte.
deletes	ul_u_long	Die Anzahl der aktuell empfangenen Zeilen, die gelöscht wurden.
ignored_deletes	ul_u_long	Die Anzahl der Zeilen im aktuellen Download, die empfangen wurden, aber in der Tabelle nicht vorhanden sind.
ignored_updates	ul_u_long	Die Anzahl der Zeilen im aktuellen Download, die empfangen wurden, und bei denen es sich um Duplikate von Zeilen handelt, die bereits in der Tabelle vorhanden sind.
inserts	ul_u_long	Die Anzahl der aktuell gesendeten eingefügten Zeilen.
truncate_deletes	ul_u_long	Die Anzahl der Zeilen, die von einem Kürzungsvorgang gelöscht wurden.
updates	ul_u_long	Die Anzahl der aktuell empfangenen Zeilen, die aktualisiert wurden.

ul_sync_stats_upload-Struktur

Gibt die Upload-Statistik des Synchronisationsdatenstroms zurück.

Syntax

```
public typedef struct ul_sync_stats_upload
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
bytes	ul_u_long	Die Anzahl der aktuell gesendeten Byte.

Mitgliedsname	Typ	Beschreibung
deletes	ul_u_long	Die Anzahl der aktuell gesendeten gelöschten Zeilen.
inserts	ul_u_long	Die Anzahl der aktuell gesendeten eingefügten Zeilen.
updates	ul_u_long	Die Anzahl der aktuell gesendeten aktualisierten Zeilen.

ul_sync_status-Struktur

Gibt die Überwachungsdaten zum Synchronisationsfortschritt zurück.

Syntax

```
public typedef struct ul_sync_status
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
current_download_row_count	ul_u_long	Die Anzahl der bis jetzt heruntergeladenen Zeilen. Diese Anzahl umfasst Duplikatzeilen, die nicht in received.inserts, received.updates oder received.deletes enthalten sind.
db_table_count	ul_u_short	Die Anzahl der Tabellen in der Datenbank.
flags	ul_u_short	Die aktuellen Synchronisationsparameter, die weitere Informationen über den aktuellen Status liefern.
info	ul_sync_info *	Ein Zeiger auf die ul_sync_info_a-Struktur.
received	ul_sync_stats_download	Downloadstatistiken.
sent	ul_sync_stats_upload	Uploadstatistiken.
sqlca	SQLCA *	Der aktive SQLCA der Verbindung.
state	ul_sync_state	Einer der unterstützten Statuswerte.
stop	ul_bool	Ein Boolescher Wert, der die Synchronisation abbricht. Der Wert TRUE gibt an, dass die Synchronisation abgebrochen wurde.
sync_table_count	ul_u_short	Die Anzahl der Tabellen, die synchronisiert werden.
sync_table_index	ul_u_short	1 .. sync_table_count

Mitgliedsname	Typ	Beschreibung
table_id	ul_u_short	Die aktuelle Tabellen-ID im Upload oder Download (Basis 1). Bei dieser Nummer werden Werte übersprungen, wenn nicht alle Tabellen synchronisiert werden. Außerdem wird diese Nummer nicht unbedingt hochgezählt.
table_name	char	Der Name der aktuellen Tabelle.
table_name_w2	ul_wchar	Der Name der aktuellen Tabelle.
total_download_row_count	ul_u_long	Die Gesamtzahl der im Download empfangenen Zeilen. Diese Anzahl umfasst Duplikatzeilen, die nicht in received.inserts, received.updates oder received.deletes enthalten sind. Dieses Feld ist nicht festgelegt, bevor die Synchronisation den UL_SYNC_STATE_RECEIVING_TABLE-Status für die erste Tabelle erreicht.
user_data	ul_void *	Die an die ULRegisterSynchronizationCallback-Methode übergebenen oder in der ul_sync_info-Struktur festgelegten Benutzerdaten.

Siehe auch

- ul_sync_state-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 92

ul_validate_data-Struktur

Speichert die Validierungsstatusinformationen für den Callback.

Syntax

```
public typedef struct ul_validate_data
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
i	ul_u_long	Parameter als Ganzzahl.
parm_count	ul_u_short	Die Anzahl der Parameter in der Struktur.
parm_type	enumeration	Die möglichen Parametertypen.
parms	struct ul_validate_data::@3	Array von Parametern.

Mitgliedsname	Typ	Beschreibung
s	char	Parameter als Zeichenfolge (dies ist kein wide-char-Datentyp)
status_id	ul_validate_status_id	Gibt an, welche Details im Validierungsverfahren berichtet werden.
stop	ul_bool	Ein boolescher Wert, der die Validierung abbricht. Der Wert TRUE gibt an, dass die Validierung abgebrochen wurde.
type	parm_type	Der Typ des gespeicherten Parameters.
user_data	ul_void *	Benutzerdefinierter Datenzeiger, der an die Validierungsroutine weitergegeben wird.

ULVF_DATABASE-Variable

Zur Validierung der Datenbank.

Syntax

```
#define ULVF_DATABASE
```

Bemerkungen

Überprüfen Sie alle Datenbankseiten mithilfe von Prüfsummen und zusätzlichen Prüfungen.

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

ULVF_EXPRESS-Variable

Für eine schnellere, allerdings weniger präzise Validierung.

Syntax

```
#define ULVF_EXPRESS
```

Bemerkungen

Dieser Parameter ändert andere angegebene Parameter.

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

ULVF_FULL_VALIDATE-Variable

Führt alle Arten von Validierungen der Datenbank durch.

Syntax

```
#define ULVF_FULL_VALIDATE
```

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

ULVF_IDX_HASH-Variable

Gibt Auskunft über die Effizienz der Index-Hashes (nur Entwicklerversion).

Syntax

```
#define ULVF_IDX_HASH
```

Bemerkungen

Prüfen Sie, ob die Zeilenanzahlen von Tabelle und Index übereinstimmen.

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

ULVF_IDX_REDUNDANT-Variable

Prüft redundante Indizes (nur Entwicklerversion).

Syntax

```
#define ULVF_IDX_REDUNDANT
```

Bemerkungen

Prüfen Sie, ob die Zeilenanzahlen von Tabelle und Index übereinstimmen.

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

ULVF_INDEX-Variable

Zur Validierung von Indizes.

Syntax

```
#define ULVF_INDEX
```

Bemerkungen

Prüfen Sie die Integrität des Indexes.

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

ULVF_TABLE-Variable

Zum Validieren von Tabellen.

Syntax

```
#define ULVF_TABLE
```

Bemerkungen

Prüfen Sie, ob die Zeilenanzahlen von Tabelle und Index übereinstimmen.

Siehe auch

- ULDatabaseManager.ValidateDatabase-Methode [UltraLite C++] auf Seite 144
- ULConnection.ValidateDatabase-Methode [UltraLite C++] auf Seite 133

UL_AS_SYNCHRONIZE-Variable

Liefert den Namen der Callback-Nachricht, mit der eine ActiveSync-Synchronisation angegeben wird

Syntax

```
#define UL_AS_SYNCHRONIZE
```

Bemerkungen

Dies gilt nur für Windows Mobile-Anwendungen, die ActiveSync verwenden.

Siehe auch

- „ActiveSync-Synchronisation einrichten“ auf Seite 71

UL_SYNC_ALL-Variable

Synchronisiert alle Tabellen in einer Datenbank, die nicht als "no sync" markiert sind, einschließlich Tabellen, die in keiner Publikation enthalten sind.

Syntax

```
#define UL_SYNC_ALL
```

Siehe auch

- ul_sync_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 96
- UL_SYNC_ALL_PUBS-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 106

UL_SYNC_ALL_PUBS-Variable

Synchronisiert alle Tabellen in einer Publikation.

Syntax

```
#define UL_SYNC_ALL_PUBS
```

Siehe auch

- ul_sync_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 96
- UL_SYNC_ALL-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 105

UL_SYNC_STATUS_FLAG_IS_BLOCKING-Variable

Definiert eine Bitgruppe im ul_sync_status.flags-Feld, um anzuzeigen, dass die Synchronisation blockiert wurde, weil auf eine Antwort des MobiLink-Servers gewartet wird.

Syntax

```
#define UL_SYNC_STATUS_FLAG_IS_BLOCKING
```

Bemerkungen

Identische Nachrichten zum Synchronisationsfortschritt werden in regelmäßigen Abständen generiert, während dies der Fall ist.

UL_ TEXT-Variable

Bereitet konstante Zeichenfolgen vor, die als Einbyte- oder breite Zeichenfolgen kompiliert werden sollen.

Syntax

```
#define UL_TEXT
```

Bemerkungen

Verwenden Sie dieses Makro, um alle konstanten Zeichenfolgen einzubeziehen, wenn Sie die Anwendung kompilieren wollen, um Unicode- und Nicht-Unicode-Repräsentationen von Zeichenfolgen zu verwenden. Dieses Makro definiert Zeichenfolgen in allen Umgebungen und auf allen Plattformen richtig.

UL_VALID_IS_ERROR-Variable

Ergibt TRUE, wenn ein bestimmter ul_validate_status_id-Wert ein Fehlerstatus ist.

Syntax

```
#define UL_VALID_IS_ERROR
```

UL_VALID_IS_INFO-Variable

Ergibt TRUE, wenn ein bestimmter ul_validate_status_id-Wert ein Informationsstatus ist.

Syntax

```
#define UL_VALID_IS_INFO
```

UltraLite C/C++-API-Referenz

Die folgende Liste enthält Beschreibungen zu einigen der häufig verwendeten API-Objekte:

- **ULDatabaseManager** Stellt Methoden für die Verwaltung von Datenbankverbindungen bereit, wie etwa CreateDatabase und OpenConnection.
- **ULConnection** Repräsentiert eine Verbindung zu einer UltraLite-Datenbank. Sie können ein oder mehrere ULConnection-Objekte erstellen.
- **ULTable** Stellt den direkten Zugriff auf Tabellen in der Datenbank bereit.
- **ULPreparedStatement, ULResultSet und ULResultSetSchema** Erstellen Dynamic SQL-Anweisungen, führen Abfragen durch, führen INSERT-, UPDATE- und DELETE-Anweisungen aus und bieten programmgesteuerte Kontrolle über Datenbank-Ergebnismengen

Header-Datei

- `ulcpp.h`

Siehe auch

- ULDatabaseManager-Klasse [UltraLite C++] auf Seite 134
- ULConnection-Klasse [UltraLite C++] auf Seite 107
- ULTable-Klasse [UltraLite C++] auf Seite 207
- ULPreparedStatement-Klasse [UltraLite C++] auf Seite 156
- ULResultSet-Klasse [UltraLite C++] auf Seite 166
- ULResultSetSchema-Klasse [UltraLite C++] auf Seite 202

ULConnection-Klasse

Repräsentiert eine Verbindung zu einer UltraLite-Datenbank.

Syntax

```
public class ULConnection
```

Mitglieder

Alle Mitglieder der ULConnection-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
CancelGetNotification-Methode	Bricht ausstehende getNotification-Aufrufe in allen Warteschlangen ab, die mit dem angegebenen Namen übereinstimmen.
ChangeEncryptionKey-Methode	Ändert den Chiffrierschlüssel für die Verschlüsselung einer UltraLite-Datenbank.
Checkpoint-Methode	Führt einen Checkpoint-Vorgang aus, der alle noch festzuschreibenden Transaktionen in der Datenbank bereinigt.
Close-Methode	Zerstört diese Verbindung und alle verbliebenen zugeordneten Objekte.
Commit-Methode	Schreibt die aktuelle Transaktion fest.
CountUploadRows-Methode	Ermittelt die Anzahl der Zeilen, die für die Synchronisation übertragen werden sollen
CreateNotificationQueue-Methode	Erstellt eine Ereignisbenachrichtigungs-Warteschlange für diese Verbindung.
DeclareEvent-Method	Deklariert ein Ereignis, für das anschließend eine Registrierung erfolgen kann und das ausgelöst werden kann.
DestroyNotificationQueue-Methode	Zerstört die angegebene Ereignisbenachrichtigungs-Warteschlange.
ExecuteScalar-Methode	Führt eine SQL SELECT-Anweisung direkt aus und gibt ein einzelnes Ergebnis zurück.
ExecuteScalarV-Methode	Führt eine SQL SELECT-Anweisung zusammen mit einer Liste von Ersetzungswerten aus.
ExecuteStatement-Methode	Führt eine SQL-Anweisung direkt aus.
GetChildObjectCount-Methode	Ruft die Anzahl der derzeit offenen untergeordneten Objekte der Verbindung ab.
GetDatabaseProperty-Methode	Ruft den Wert einer Datenbankeigenschaft ab.

Name	Beschreibung
GetDatabasePropertyInt-Methode	Ruft den Ganzzahlwert einer Datenbankeigenschaft ab.
GetDatabaseSchema-Methode	Gibt einen Objektverweis zurück, der verwendet wird, um das Schema der Datenbank abzufragen.
GetLastDownloadTime-Methode	Fragt den letzten Zeitpunkt ab, zu dem eine bestimmte Publikation heruntergeladen wurde
GetLastError-Methode	Gibt die Fehlerinformationen im Zusammenhang mit dem letzten Aufruf zurück.
GetLastIdentity-Method	Ruft den @@identity-Wert ab.
GetNotification-Methode	Liest eine Ereignisbenachrichtigung.
GetNotificationParameter-Methode	Ruft einen Parameter für die Ereignisbenachrichtigung ab, die gerade von der GetNotification-Methode gelesen wurde.
GetSqlca-Methode	Ruft den Kommunikationsbereich ab, dem diese Verbindung zugeordnet ist.
GetSyncResult-Methode	Ruft das Ergebnis der letzten Synchronisation ab.
GetUserPointer-Methode	Ruft den zuletzt von der SetUserPointer-Methode gesetzen Zeigerwert ab.
GlobalAutoincUsage-Methode	Ruft den Prozentwert der Standardwerte ab, die in allen Spalten verwendet werden, die global autoincrement-Standardwerte haben.
GrantConnectTo-Methode	Erteilt einer vorhandenen Benutzer-ID mit dem betreffenden Kennwort Zugriff auf eine UltraLite-Datenbank.
InitSyncInfo-Methode	Initialisiert die Synchronisationsinformationsstruktur.
OpenTable-Methode	Öffnet eine Tabelle.
PrepareStatement-Methode	Bereitet eine SQL-Anweisung vor.
RegisterForEvent-Methode	Registriert eine Warteschlange, um Benachrichtigungen für ein Ereignis zu empfangen, oder macht diese Registrierung rückgängig.
ResetLastDownloadTime-Methode	Diese Methode setzt die Zeit des letzten Downloads einer Publikation zurück, sodass die Anwendung bereits heruntergeladene Daten erneut synchronisiert.

Name	Beschreibung
RevokeConnectFrom-Methode	Entzieht einer Benutzer-ID die Zugriffsberechtigung auf eine UltraLite-Datenbank
Rollback-Methode	Setzt die aktuelle Transaktion zurück.
RollbackPartialDownload-Methode	Setzt die Änderungen einer fehlgeschlagenen Synchronisation zurück.
SendNotification-Methode	Sendet eine Nachricht an alle Warteschlangen, die mit dem angegebenen Namen übereinstimmen.
SetDatabaseOption-Methode	Legt die angegebene Datenbankoption fest.
SetDatabaseOptionInt-Methode	Legt eine Datenbankoption fest.
SetSynchronizationCallback-Methode	Legt den Callback fest, der während einer Synchronisation aufgerufen werden soll.
SetSyncInfo-Methode	Erstellt ein Synchronisationsprofil unter Verwendung des angegebenen Namens basierend auf der angegebenen ul_sync_info-Struktur.
SetUserPointer-Methode	Legt einen beliebigen Zeigerwert in der Verbindung für die Verwendung durch die aufrufende Anwendung fest.
StartSynchronizationDelete-Methode	Legt START SYNCHRONIZATION DELETE für diese Verbindung fest.
StopSynchronizationDelete-Methode	Legt STOP SYNCHRONIZATION DELETE für diese Verbindung fest.
Synchronize-Methode	Initiiert die Synchronisation in einer UltraLite-Anwendung
SynchronizeFromProfile-Methode	Synchronisiert die Datenbank unter Verwendung der angegebenen Parameter profile und merge.
TriggerEvent-Methode	Löst ein benutzerdefiniertes Ereignis aus und sendet eine Benachrichtigung an alle registrierten Warteschlangen.
ValidateDatabase-Methode	Validiert die Datenbank in dieser Verbindung.

CancelGetNotification-Methode

Bricht ausstehende getNotification-Aufrufe in allen Warteschlangen ab, die mit dem angegebenen Namen übereinstimmen.

Syntax

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

Parameter

- **queueName** Der Name der Warteschlange.

Rückgabe

Die Anzahl der betroffenen Warteschlangen (nicht notwendigerweise die Anzahl der blockierten Lesevorgänge).

ChangeEncryptionKey-Methode

Ändert den Chiffrierschlüssel für die Verschlüsselung einer UltraLite-Datenbank.

Syntax

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

Parameter

- **newKey** Der neue Chiffrierschlüssel für die Datenbank.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Anwendungen, die diese Methode aufrufen, müssen zunächst sicherstellen, dass der Benutzer entweder die Datenbank synchronisiert oder eine zuverlässige Sicherungskopie der Datenbank erstellt hat. Die zuverlässige Sicherung der Datenbank ist wichtig, da die ChangeEncryptionKey-Methode vollständig ausgeführt werden muss. Wenn Sie den Datenbank-Chiffrierschlüssel ändern, wird jede Zeile in der Datenbank zuerst mit dem alten Schlüssel entschlüsselt und dann mit dem neueren Schlüssel verschlüsselt, bevor die Zeile neu geschrieben wird. Dieser Vorgang kann nicht rückgängig gemacht werden. Wenn der Änderungsvorgang der Verschlüsselung nicht abgeschlossen wird, hat die Datenbank einen ungültigen Status und Sie können nicht mehr auf sie zugreifen.

Checkpoint-Methode

Führt einen Checkpoint-Vorgang aus, der alle noch festzuschreibenden Transaktionen in der Datenbank bereinigt.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Aktuell ausgeführte Transaktionen werden durch Aufrufen der Checkpoint-Methode nicht festgeschrieben. Diese Methode wird in Verbindung mit dem Verzögern automatischer Transaktions-Checkpoints (mithilfe des **commit_flush**-Verbindungsparameters) als Performance-Verbesserung eingesetzt.

Die Checkpoint-Methode stellt sicher, dass alle noch festzuschreibenden Transaktionen in die Datenbank geschrieben wurden.

Close-Methode

Zerstört diese Verbindung und alle verbliebenen zugeordneten Objekte.

Syntax

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

Parameter

- **error** Ein optionales ULError-Objekt für die Aufnahme der Fehlerinformationen.

Commit-Methode

Schreibt die aktuelle Transaktion fest.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

CountUploadRows-Methode

Ermittelt die Anzahl der Zeilen, die für die Synchronisation übertragen werden sollen

Syntax

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

Parameter

- **pubList** Eine Zeichenfolge mit einer kommagetrennten Liste von Publikationen, die geprüft werden sollen. Eine leere Zeichenfolge (der UL_SYNC_ALL-Makro) bedeutet alle Tabellen, außer den als "no sync" markierten. Eine Zeichenfolge, die nur ein Sternchen enthält (der UL_SYNC_ALL_PUBS-Makro), bedeutet alle Tabellen, die in einer Publikation referenziert werden.

Einige Tabellen sind in keiner Publikation referenziert und werden daher nicht einbezogen, wenn dieser Wert "*" ist.

- **threshold** Ermittelt die maximale Anzahl der zu zählenden Zeilen, wodurch die Zeitdauer des Aufrufs begrenzt wird. Der Schwellenwert 0 entspricht keiner Beschränkung (d.h., es werden alle zu synchronisierenden Zeilen gezählt) und der Schwellenwert 1 kann verwendet werden, um schnell zu ermitteln, ob Zeilen synchronisiert werden müssen.

Rückgabe

Die Anzahl der Zeilen, die synchronisiert werden müssen, entweder in einer angegebenen Gruppe von Publikationen oder in der gesamten Datenbank.

Bemerkungen

Verwenden Sie diese Methode, um Benutzer zur Synchronisation aufzufordern oder um festzulegen, wann die automatische Hintergrundsynchronisation erfolgen soll.

Der folgende Aufruf überprüft die gesamte Datenbank auf die Gesamtzahl der zu synchronisierenden Zeilen:

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

Folgender Aufruf überprüft die Publikationen PUB1 und PUB2 auf ein Maximum von 1000 Zeilen:

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

Der folgende Aufruf prüft, ob in den Publikationen PUB1 und PUB2 Zeilen synchronisiert werden sollen:

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

CreateNotificationQueue-Methode

Erstellt eine Ereignisbenachrichtigungs-Warteschlange für diese Verbindung.

Syntax

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

Parameter

- **name** Der Name der neuen Warteschlange.
- **parameters** Reserviert. Auf NULL setzen.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Der Bereich von Warteschlangennamen gilt für die jeweilige Verbindung, daher können verschiedene Verbindungen Warteschlangen mit demselben Namen erstellen. Wenn eine Ereignisbenachrichtigung

gesendet wird, empfangen alle Warteschlangen in der Datenbank mit einem übereinstimmenden Namen eine (separate Instanz der) Benachrichtigung. Namen reagieren nicht auf Groß- und Kleinschreibung. Beim Aufrufen der RegisterForEvent-Methode wird für jede Verbindung eine Standard-Warteschlange erstellt, falls keine Warteschlange angegeben ist. Dieser Abfruf schlägt mit einem Fehler fehl, wenn der Name bereits vorhanden oder nicht gültig ist.

Siehe auch

- ULConnection.RegisterForEvent-Methode [UltraLite C++] auf Seite 125

DeclareEvent-Method

Deklariert ein Ereignis, für das anschließend eine Registrierung erfolgen kann und das ausgelöst werden kann.

Syntax

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

Parameter

- **eventName** Der Name für ein neues benutzerdefiniertes Ereignis.

Rückgabe

TRUE, wenn das Ereignis erfolgreich deklariert wurde, und FALSE, wenn der Name bereits verwendet wird oder ungültig ist.

Bemerkungen

In UltraLite gibt es vordefinierte Systemereignisse, die von Vorgängen in der Datenbank oder in der Umgebung ausgelöst werden. Diese Methode deklariert benutzerdefinierte Ereignisse. Benutzerdefinierte Ereignisse werden mit der TriggerEvent-Methode ausgelöst. Der Ereignisname muss eindeutig sein. Namen berücksichtigen nicht die Groß- und Kleinschreibung.

Siehe auch

- ULConnection.TriggerEvent-Methode [UltraLite C++] auf Seite 133

DestroyNotificationQueue-Methode

Zerstört die angegebene Ereignisbenachrichtigungs-Warteschlange.

Syntax

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

Parameter

- **name** Der Name der zu zerstörenden Warteschlange.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Eine Warnung wird signalisiert, wenn ungelesene Benachrichtigungen in der Warteschlange verbleiben. Ungelesene Benachrichtigungen werden verworfen. Die Standard-Ereigniswarteschlange einer Verbindung, falls erstellt, wird vernichtet, wenn die Verbindung geschlossen wird.

ExecuteScalar-Methode

Führt eine SQL SELECT-Anweisung direkt aus und gibt ein einzelnes Ergebnis zurück.

Syntax

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

Parameter

- **dstPtr** Ein Zeiger auf eine Variable des erforderlichen Datentyps, um den Wert aufzunehmen.
- **dstSize** Die Größe der Variablen für den Empfang des Werts, falls zutreffend.
- **dstType** Der Typ des abzurufenden Werts. Dieser Wert muss mit dem Variablentyp übereinstimmen.
- **sql** Die SELECT-Anweisung, optional mit '?'- Parametern.
- **...** String (char *)-Parameterwerte zum Ersetzen.

Rückgabe

TRUE, wenn die Abfrage erfolgreich ausgeführt und ein Wert abgerufen wurde, und FALSE, wenn kein Wert abgerufen wurde. Prüfen Sie den SQLCODE-Fehlercode, um zu ermitteln, warum FALSE zurückgegeben wurde. Der ausgewählte Wert ist NULL, wenn keine Warnung oder ein Fehler (SQLE_NOERROR) angegeben wird.

Bemerkungen

Der dstPtr-Wert muss auf eine Variable des richtigen Typs zeigen, entsprechend dem dstType-Wert. Der dstSize-Parameter ist nur für Werte mit variabler Größe erforderlich, z.B. für Zeichenfolgen und Binärdaten. Ansonsten wird er ignoriert. Die variable Liste der Parameterwerte muss den Parametern in der Anweisung entsprechen und es wird angenommen, dass alle Werte Zeichenfolgen sind. (Intern wandelt UltraLite die Parameterwerte wie für die Anweisung erforderlich um.)

Die folgenden Typen werden unterstützt:

- **UL_TYPE_BIT/UL_TYPE_TINY** Verwenden Sie den Variablentyp ul_byte (8 Bit, ohne Vorzeichen).
- **UL_TYPE_U_SHORT/UL_TYPE_S_SHORT** Verwenden Sie den Variablentyp ul_u_short/ ul_s_short (16 Bit).
- **UL_TYPE_U_LONG/UL_TYPE_S_LONG** Verwenden Sie den Variablentyp ul_u_long/ul_s_long (32 Bit).
- **UL_TYPE_U_BIG/UL_TYPE_S_BIG** Verwenden Sie den Variablentyp ul_u_big/ul_s_big (64 Bit).
- **UL_TYPE_DOUBLE** Verwenden Sie den Variablentyp ul_double (double).
- **UL_TYPE_REAL** Verwenden Sie den Variablentyp ul_real (float).
- **UL_TYPE_BINARY** Verwenden Sie den Variablentyp und geben Sie ul_binary **dstSize** (wie in GetBinary()) an.
- **UL_TYPE_TIMESTAMP_STRUCT** Verwenden Sie den Variablentyp DECL_DATETIME.
- **UL_TYPE_CHAR** Verwenden Sie den Variablentyp char [] (Zeichenpuffer) und legen Sie **dstSize** mit der Größe des Puffers fest (wie in GetString()).
- **UL_TYPE_WCHAR** Verwenden Sie den Variablentyp ul_wchar [] (ein breiter Zeichenpuffer) und legen Sie **dstSize** mit der Größe des Puffers fest (wie in GetString()).
- **UL_TYPE_TCHAR** Entspricht UL_TYPE_CHAR oder UL_TYPE_WCHAR, abhängig davon, welche Version der Methode aufgerufen wird.

Das folgende Beispiel zeigt den Aufruf einer Ganzzahl:

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

Das folgende Beispiel zeigt den Aufruf einer Zeichenfolge:

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

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

ExecuteScalarV-Methode

Führt eine SQL SELECT-Anweisung zusammen mit einer Liste von Ersetzungswerten aus.

Syntax

```
public virtual bool ExecuteScalarV(
    void * dstPtr,
```

```
    size_t dstSize,
    ul_column_storage_type dstType,
    const char * sql,
    va_list args
)
```

Parameter

- **dstPtr** Ein Zeiger auf eine Variable des erforderlichen Datentyps, um den Wert aufzunehmen.
- **dstSize** Die Größe der Variablen für den Empfang des Werts, falls zutreffend.
- **dstType** Der Typ des abzurufenden Werts. Dieser Wert muss mit dem Variablentyp übereinstimmen.
- **sql** Die SELECT-Anweisung, optional mit '?'- Parametern.
- **args** Eine Liste von Zeichenfolgenwerten (char *) zum Ersetzen.

Rückgabe

TRUE, wenn die Abfrage erfolgreich ausgeführt und ein Wert abgerufen wurde, und FALSE, wenn kein Wert abgerufen wurde. Prüfen Sie den SQLCODE-Fehlercode, um zu ermitteln, warum FALSE zurückgegeben wurde. Der ausgewählte Wert ist NULL, wenn keine Warnung oder ein Fehler (SQLE_NOERROR) angegeben wird.

Bemerkungen

Der dstPtr-Wert muss auf eine Variable des richtigen Typs zeigen, entsprechend dem dstType-Wert. Der dstSize-Parameter ist nur für Werte mit variabler Größe erforderlich, z.B. für Zeichenfolgen und Binärdaten. Ansonsten wird er ignoriert. Die variable Liste der Parameterwerte muss den Parametern in der Anweisung entsprechen und es wird angenommen, dass alle Werte Zeichenfolgen sind. (Intern wandelt UltraLite die Parameterwerte wie für die Anweisung erforderlich um.)

Die folgenden Typen werden unterstützt:

- **UL_TYPE_BIT/UL_TYPE_TINY** Verwenden Sie den Variablentyp ul_byte (8 Bit, ohne Vorzeichen).
- **UL_TYPE_U_SHORT/UL_TYPE_S_SHORT** Verwenden Sie den Variablentyp ul_u_short/ ul_s_short (16 Bit).
- **UL_TYPE_U_LONG/UL_TYPE_S_LONG** Verwenden Sie den Variablentyp ul_u_long/ul_s_long (32 Bit).
- **UL_TYPE_U_BIG/UL_TYPE_S_BIG** Verwenden Sie den Variablentyp ul_u_big/ul_s_big (64 Bit).
- **UL_TYPE_DOUBLE** Verwenden Sie den Variablentyp ul_double (double).
- **UL_TYPE_REAL** Verwenden Sie den Variablentyp ul_real (float).
- **UL_TYPE_BINARY** Verwenden Sie den Variablentyp und geben Sie ul_binary **dstSize** (wie in GetBinary()) an.

- **UL_TYPE_TIMESTAMP_STRUCT** Verwenden Sie den Variablentyp DECL_DATETIME.
- **UL_TYPE_CHAR** Verwenden Sie den Variablentyp char [] (Zeichenpuffer) und legen Sie **dstSize** mit der Größe des Puffers fest (wie in GetString()).
- **UL_TYPE_WCHAR** Verwenden Sie den Variablentyp ul_wchar [] (ein breiter Zeichenpuffer) und legen Sie **dstSize** mit der Größe des Puffers fest (wie in GetString()).
- **UL_TYPE_TCHAR** Entspricht UL_TYPE_CHAR oder UL_TYPE_WCHAR, abhängig davon, welche Version der Methode aufgerufen wird.

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

ExecuteStatement-Methode

Führt eine SQL-Anweisung direkt aus.

Syntax

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

Parameter

- **sql** Das auszuführende SQL-Skript.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Verwenden Sie diese Methode, um eine SELECT-Anweisung direkt auszuführen und ein einzelnes Ergebnis abzurufen.

Verwenden Sie die PrepareStatement-Methode wiederholt mit variablen Parametern oder zum Abrufen mehrerer Ergebnisse.

Siehe auch

- ULConnection.PrepareStatement-Methode [UltraLite C++] auf Seite 125

GetChildObjectCount-Methode

Ruft die Anzahl der derzeit offenen untergeordneten Objekte der Verbindung ab.

Syntax

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

Rückgabe

Die Anzahl der derzeit offenen untergeordneten Objekte.

Bemerkungen

Diese Methode kann zur Erkennung von Objektlücken verwendet werden.

GetDatabaseProperty-Methode

Ruft den Wert einer Datenbankeigenschaft ab.

Syntax

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

Parameter

- **propName** Der Name der angeforderten Eigenschaft.

Rückgabe

Bei erfolgreicher Ausführung wird ein Zeiger auf einen Zeichenfolgenpuffer zurückgegeben, der den Datenbankeigenschaftswert enthält, ansonsten NULL.

Bemerkungen

Der zurückgegebene Wert verweist auf einen statischen Puffer, dessen Inhalt durch einen nachfolgenden UltraLite-Aufruf geändert werden kann, daher müssen Sie eine Kopie des Werts herstellen, wenn Sie ihn speichern wollen.

Siehe auch

- „UltraLite-Datenbankeigenschaften“ [*UltraLite - Datenbankverwaltung*]

Beispiel

Das folgende Beispiel zeigt, wie Sie den Wert der CharSet-Datenbankeigenschaft abrufen.

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

GetDatabasePropertyInt-Methode

Ruft den Ganzzahlwert einer Datenbankeigenschaft ab.

Syntax

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

Parameter

- **propName** Der Name der angeforderten Eigenschaft.

Rückgabe

Bei erfolgreicher Ausführung wird der Ganzzahlwert der Eigenschaft zurückgegeben, ansonsten 0.

Siehe auch

- „UltraLite-Datenbankeigenschaften“ [*UltraLite - Datenbankverwaltung*]

Beispiel

Das folgende Beispiel zeigt, wie Sie den Wert der ConnCount-Datenbankeigenschaft abrufen.

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

GetDatabaseSchema-Methode

Gibt einen Objektverweis zurück, der verwendet wird, um das Schema der Datenbank abzufragen.

Syntax

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

Rückgabe

Ein ULDatabaseSchema-Objekt, das für die Abfrage des Schemas aus der Datenbank verwendet wird.

GetLastDownloadTime-Methode

Fragt den letzten Zeitpunkt ab, zu dem eine bestimmte Publikation heruntergeladen wurde

Syntax

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

Parameter

- **publication** Der Publikationsname.
- **value** Ein Zeiger auf die DECL_DATETIME-Struktur, die gefüllt werden soll. Der Wert 1. Januar 1990 bedeutet, dass entweder die Publikation noch nicht synchronisiert oder die Zeit zurückgesetzt wurde.

Rückgabe

TRUE, wenn der Wert erfolgreich mit dem letzten Downloadzeitpunkt der angegebenen Publikation gefüllt wurde, ansonsten FALSE.

Bemerkungen

Der folgende Aufruf füllt die dt-Struktur mit dem Datum und der Uhrzeit des letzten Downloads der Publikation "pub1":

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

GetLastError-Methode

Gibt die Fehlerinformationen im Zusammenhang mit dem letzten Aufruf zurück.

Syntax

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

Rückgabe

Ein Zeiger auf das ULError-Objekt mit Informationen zum letzten Aufruf.

Bemerkungen

Das Fehlerobjekt, dessen Adresse zurückgegeben wird, bleibt gültig, solange die Verbindung geöffnet ist, wird aber für nachfolgende Aufrufe nicht automatisch aktualisiert. Sie müssen GetLastError aufrufen, um aktualisierte Statusinformationen abzurufen.

Siehe auch

- ULError-Klasse [UltraLite C++] auf Seite 148

GetLastIdentity-Method

Ruft den @@identity-Wert ab.

Syntax

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

Rückgabe

Der letzte eingefügte Wert in eine Autoincrement- oder Global-Autoincrement-Spalte.

Bemerkungen

Dieser Wert ist der letzte eingefügte Wert in eine Autoincrement- oder Global-Autoincrement-Spalte für die Datenbank. Dieser Wert wird nicht aufgezeichnet, wenn die Datenbank heruntergefahren wird, daher gibt der Aufruf dieser Methode, bevor Autoincrement-Werte eingefügt wurden, 0 zurück.

> **Hinweis**
> Der letzte Wert wurde möglicherweise über eine andere Verbindung eingefügt.

GetNotification-Methode

Liest eine Ereignisbenachrichtigung.

Syntax

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

Parameter

- **queueName** Die zu lesende Warteschlange oder NULL für die Standard-Verbindungswarteschlange.
- **waitms** Die abzuwartende Zeitspanne (Blockierung, in Millisekunden) vor der Rückgabe.

Rückgabe

Der Name des gelesenen Ereignisses oder bei Fehler NULL.

Bemerkungen

Dieser Aufruf wird blockiert, bis eine Benachrichtigung empfangen wird oder die angegebene Wartezeit abgelaufen ist. Für einen unbegrenzten Wartezeitraum setzen Sie den waitms-Parameter auf UL_READ_WAIT_INFINITE. Um den Wartezustand zu beenden, senden Sie eine weitere Benachrichtigung an die angegebene Warteschlange oder verwenden Sie die CancelGetNotification-Methode. Verwenden Sie nach dem Lesen einer Benachrichtigung die GetNotificationParameter-Methode, um zusätzliche Parameter nach Namen abzurufen.

Siehe auch

- ULConnection.CancelGetNotification-Methode [UltraLite C++] auf Seite 110
- ULConnection.GetNotificationParameter-Methode [UltraLite C++] auf Seite 122

GetNotificationParameter-Methode

Ruft einen Parameter für die Ereignisbenachrichtigung ab, die gerade von der GetNotification-Methode gelesen wurde.

Syntax

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

Parameter

- **queueName** Zu lesende Warteschlange oder NULL für Standard-Verbindungswarteschlange.
- **parameterName** Der Name des Parameters, der gelesen werden soll (oder "*").

Rückgabe

Der Parameterwert oder NULL bei einem Fehler.

Bemerkungen

Es sind nur die Parameter aus der zuletzt gelesenen Benachrichtigung in der angegebenen Warteschlange verfügbar. Parameter werden nach Namen abgerufen. Mit dem Parameternamen "*" wird die komplette Parameterzeichenfolge abgerufen.

Siehe auch

- ULConnection.GetNotification-Methode [UltraLite C++] auf Seite 121

GetSqlca-Methode

Ruft den Kommunikationsbereich ab, dem diese Verbindung zugeordnet ist.

Syntax

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

Rückgabe

Ein Zeiger auf das SQLCA-Objekt für diese Verbindung.

GetSyncResult-Methode

Ruft das Ergebnis der letzten Synchronisation ab.

Syntax

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

Parameter

- **syncResult** Ein Zeiger auf die ul_sync_result-Struktur, die gefüllt werden soll.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Siehe auch

- ul_sync_result-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 99

GetUserPointer-Methode

Ruft den zuletzt von der SetUserPointer-Methode gesetzen Zeigerwert ab.

Syntax

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

Siehe auch

- ULConnection.SetUserPointer-Methode [UltraLite C++] auf Seite 130

GlobalAutoincUsage-Methode

Ruft den Prozentwert der Standardwerte ab, die in allen Spalten verwendet werden, die global autoincrement-Standardwerte haben.

Syntax

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

Rückgabe

Der Prozentwert des Global-Autoincrement-Werts, der vom Zähler verwendet wird.

Bemerkungen

Wenn die Datenbank mehr als eine Spalte mit diesem Standardwert enthält, wird dieser Wert für alle Spalten berechnet und das Maximum wird zurückgegeben. Der Rückgabewert 99 weist z.B. darauf hin, dass zumindest für eine der Spalten sehr wenige Standardwerte übrig sind.

GrantConnectTo-Methode

Erteilt einer vorhandenen Benutzer-ID mit dem betreffenden Kennwort Zugriff auf eine UltraLite-Datenbank.

Syntax

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

Parameter

- **uid** Ein Zeichen-Array, das die Benutzer-ID enthält. Die maximale Länge beträgt 31 Zeichen.
- **pwd** Ein Zeichen-Array, das das Kennwort für die Benutzer-ID enthält.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Diese Methode aktualisiert das Kennwort für einen vorhandenen Benutzer, wenn Sie eine vorhandene Benutzer-ID angeben.

Siehe auch

- ULConnection.RevokeConnectFrom-Methode [UltraLite C++] auf Seite 127

InitSyncInfo-Methode

Initialisiert die Synchronisationsinformationsstruktur.

Syntax

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

Parameter

- **info** Ein Zeiger auf die ul_sync_info-Struktur, die die Synchronisationsparameter enthält.

Bemerkungen

Rufen Sie diese Methode auf, bevor Sie die Werte von Feldern in der ul_sync_info-Struktur festlegen.

OpenTable-Methode

Öffnet eine Tabelle.

Syntax

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

Parameter

- **tableName** Der Name der Tabelle, die geöffnet werden soll
- **indexName** Der Name des Indexes, mit dem die Tabelle geöffnet werden soll. Übergeben Sie NULL, um die Tabelle nach Primärschlüssel zu öffnen, und lassen Sie die Zeichenfolge leer, um sie ungeordnet zu öffnen.

Rückgabe

Wenn der Aufruf erfolgreich ist, wird das ULTable-Objekt zurückgegeben, ansonsten NULL.

Bemerkungen

Die Cursorposition wird vor die erste Zeile gesetzt, wenn die Anwendung eine Tabelle erstmals öffnet.

PrepareStatement-Methode

Bereitet eine SQL-Anweisung vor.

Syntax

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

Parameter

- **sql** Die vorzubereitende SQL-Anweisung.

Rückgabe

Bei Erfolg wird das ULPreparedStatement-Objekt zurückgegeben, ansonsten NULL.

RegisterForEvent-Methode

Registriert eine Warteschlange, um Benachrichtigungen für ein Ereignis zu empfangen, oder macht diese Registrierung rückgängig.

Syntax

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

Parameter

- **eventName** Das system- oder benutzerdefinierte Ereignis, für das eine Registrierung erfolgen soll.
- **objectName** Das Objekt, für das das Ereignis gilt (z.B. ein Tabellenname).
- **queueName** NULL bedeutet, dass die Standard-Verbindungswarteschlange verwendet werden soll.
- **register_not_unreg** Setzen Sie den Wert zum Registrieren auf TRUE oder zum Deregistrieren auf FALSE.

Rückgabe

TRUE, wenn die Registrierung erfolgreich war, und FALSE, wenn die Warteschlange oder das Ereignis nicht vorhanden ist.

Bemerkungen

Wenn kein Warteschlangenname geliefert wird, wird die Standard-Verbindungswarteschlange angenommen und ggf. erstellt. Bei bestimmten Systemereignissen können Sie den Namen eines Objekts angeben, für das das Ereignis gilt. Das TableModified-Ereignis z.B. kann den Tabellennamen spezifizieren. Im Gegensatz zur SendNotification-Methode empfängt nur die jeweilige registrierte Warteschlange Benachrichtigungen des Ereignisses. Andere Warteschlangen mit demselben Namen auf anderen Verbindungen erhalten keine Benachrichtigungen, es sei denn, sie sind ebenfalls explizit registriert.

Die vordefinierten Systemereignisse sind:

- **TableModified** Wird ausgelöst, wenn Zeilen in einer Tabelle eingefügt, aktualisiert oder gelöscht werden. Eine Benachrichtigung wird pro Anforderung gesendet, unbeschadet der Anzahl der durch die Anforderung betroffenen Zeilen. Der Parameter object_name gibt die zu überwachende Tabelle an. Der Wert "*" steht für alle Tabellen in der Datenbank. Dieses Ereignis hat einen Parameter namens "table_name", dessen Wert der Name der geänderten Tabelle ist.
- **Commit** Wird ausgelöst, nachdem ein Festschreibvorgang abgeschlossen ist. Dieses Ereignis hat keine Parameter.
- **SyncComplete** Wird ausgelöst, nachdem die Synchronisation abgeschlossen ist. Dieses Ereignis hat keine Parameter.

ResetLastDownloadTime-Methode

Diese Methode setzt die Zeit des letzten Downloads einer Publikation zurück, sodass die Anwendung bereits heruntergeladene Daten erneut synchronisiert.

Syntax

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

Parameter

- **pubList** Eine Zeichenfolge mit einer kommagetrennten Liste von Publikationen, die zurückgesetzt werden sollen. Eine leere Zeichenfolge bedeutet alle Tabellen, außer den als "no sync" markierten. Eine Zeichenfolge, die nur ein Sternchen ("*") enthält, bedeutet alle Publikationen. Einige Tabellen sind in keiner Publikation referenziert und werden daher nicht einbezogen, wenn dieser Wert "*" ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Der folgende Methodenaufruf setzt die Zeit des letzten Downloads für alle Tabellen zurück:

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

RevokeConnectFrom-Methode

Entzieht einer Benutzer-ID die Zugriffsberechtigung auf eine UltraLite-Datenbank

Syntax

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

Parameter

- **uid** Ein Zeichen-Array, das die Benutzer-ID speichert, die vom Zugriff auf die Datenbank ausgeschlossen werden soll.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Rollback-Methode

Setzt die aktuelle Transaktion zurück.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

RollbackPartialDownload-Methode

Setzt die Änderungen einer fehlgeschlagenen Synchronisation zurück.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Wenn Sie wiederaufnehmbare Downloads (Synchronisation mit aktivierter keep-partial-download-Option) verwenden und während der Downloadphase der Synchronisation ein Kommunikationsfehler auftritt, bewahrt UltraLite die heruntergeladenen Änderungen auf (sodass die Synchronisation an der Stelle wieder aufgenommen werden kann, an der sie unterbrochen wurde). Verwenden Sie diese Methode, um diesen Teil-Download zu verwerfen, wenn Sie nicht mehr versuchen möchten, die Synchronisation wieder aufzunehmen.

Diese Methode hat nur dann Auswirkungen, wenn Sie wiederaufnehmbare Downloads verwenden.

SendNotification-Methode

Sendet eine Nachricht an alle Warteschlangen, die mit dem angegebenen Namen übereinstimmen.

Syntax

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

Parameter

- **queueName** Der Zielwarteschlangenname (oder "*").
- **eventName** Die Identität für die Benachrichtigung.
- **parameters** Optionale Parameteroptionsliste.

Rückgabe

Anzahl der gesendeten Benachrichtigungen (die Anzahl der übereinstimmenden Warteschlangen).

Bemerkungen

Dies umfasst jede Warteschlange in der aktuellen Verbindung. Dieser Aufruf bewirkt keine Blockierung. Verwenden Sie den speziellen Warteschlangennamen "*", um Benachrichtigungen an alle Warteschlangen zu senden. Der angegebene Ereignisname muss nicht mit einem system- oder benutzerdefinierten Ereignis übereinstimmen. Er wird einfach weitergereicht, um die Benachrichtigung beim Lesen zu identifizieren, und ist nur für Absender und Empfänger von Bedeutung.

Der Wert **parameters** wird in Form einer durch Semikolons getrennten Optionsliste aus Name=Wert-Paaren übergeben. Nachdem die Nachricht gelesen wurde, werden die Parameterwerte mit der GetNotificationParameter-Methode gelesen.

Siehe auch

- ULConnection.GetNotificationParameter-Methode [UltraLite C++] auf Seite 122

SetDatabaseOption-Methode

Legt die angegebene Datenbankoption fest.

Syntax

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

Parameter

- **optName** Der Name der festzulegenden Option.
- **value** Der neue Wert für die Option.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Siehe auch

- „UltraLite-Datenbankoptionen“ [*UltraLite - Datenbankverwaltung*]

SetDatabaseOptionInt-Methode

Legt eine Datenbankoption fest.

Syntax

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

Parameter

- **optName** Der Name der festzulegenden Option.
- **value** Der neue Wert für die Option.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetSynchronizationCallback-Methode

Legt den Callback fest, der während einer Synchronisation aufgerufen werden soll.

Syntax

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

Parameter

- **callback** Der ul_sync_observer_fn-Callback.
- **userData** An den Callback übergebene Benutzerkontextinformationen.

SetSyncInfo-Methode

Erstellt ein Synchronisationsprofil unter Verwendung des angegebenen Namens basierend auf der angegebenen ul_sync_info-Struktur.

Syntax

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

Parameter

- **profileName** Der Name des Synchronisationsprofils.
- **info** Ein Zeiger auf die ul_sync_info-Struktur, die die Synchronisationsparameter enthält.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Das Synchronisationsprofil ersetzt alle vorherigen Profile mit demselben Namen. Das benannte Profil wird durch die Angabe eines Nullzeigers für die Struktur gelöscht.

SetUserPointer-Methode

Legt einen beliebigen Zeigerwert in der Verbindung für die Verwendung durch die aufrufende Anwendung fest.

Syntax

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

Rückgabe

Der zuvor eingestellte Zeigerwert.

Bemerkungen

Dies kann verwendet werden, um Anwendungsdaten der Verbindung zuzuordnen.

StartSynchronizationDelete-Methode

Legt START SYNCHRONIZATION DELETE für diese Verbindung fest.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

StopSynchronizationDelete-Methode

Legt STOP SYNCHRONIZATION DELETE für diese Verbindung fest.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Synchronize-Methode

Initiiert die Synchronisation in einer UltraLite-Anwendung

Syntax

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

Parameter

- **info** Ein Zeiger auf die ul_sync_info-Struktur, die die Synchronisationsparameter enthält.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Diese Methode startet die Synchronisation mit dem MobiLink-Server. Diese Methode gibt erst dann Ergebnisse zurück, wenn die Synchronisation abgeschlossen ist. Zusätzliche Threads auf getrennten Verbindungen können jedoch während der Synchronisation weiterhin auf die Datenbank zugreifen.

Bevor Sie diese Methode aufrufen, aktivieren Sie mit Methoden der ULDatabaseManager-Klasse das entsprechende Protokoll und die entsprechende Verschlüsselung. Wenn Sie beispielsweise "HTTP" verwenden, rufen Sie die ULDatabaseManager.EnableHttpSynchronization-Methode auf.

```
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 );
```

Siehe auch

- ULDatabaseManager.EnableHttpSynchronization-Methode [UltraLite C++] auf Seite 139
- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

SynchronizeFromProfile-Methode

Synchronisiert die Datenbank unter Verwendung der angegebenen Parameter profile und merge.

Syntax

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

Parameter

- **profileName** Der Name des zu synchronisierenden Profils.
- **mergeParms** Merge-Parameter für die Synchronisation.
- **observer** Der Observer-Callback, an den Statusaktualisierungen gesendet werden sollen.
- **userData** An den Callback übergebene Benutzerkontextdaten.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Diese Methode ist identisch mit dem Ausführen der SYNCHRONIZE-Anweisung.

Siehe auch

- ULConnection.Synchronize-Methode [UltraLite C++] auf Seite 131
- „SYNCHRONIZE-Anweisung [UltraLite]“ [*UltraLite - Datenbankverwaltung*]

TriggerEvent-Methode

Löst ein benutzerdefiniertes Ereignis aus und sendet eine Benachrichtigung an alle registrierten Warteschlangen.

Syntax

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

Parameter

- **eventName** Der Name des Systemereignisses oder benutzerdefinierten Ereignisses, das ausgelöst werden soll.
- **parameters** Optionale Parameteroptionsliste.

Rückgabe

Die Anzahl der gesendeten Ereignisbenachrichtigungen.

Bemerkungen

Der Wert **parameters** wird in Form einer durch Semikolons getrennten Optionsliste aus Name=Wert-Paaren übergeben. Nachdem die Nachricht gelesen wurde, werden die Parameterwerte mit GetNotificationParameter gelesen().

Siehe auch

- ULConnection.GetNotificationParameter-Methode [UltraLite C++] auf Seite 122

ValidateDatabase-Methode

Validiert die Datenbank in dieser Verbindung.

Syntax

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

Parameter

- **flags** Parameter, die die Art der Validierung steuern. Siehe untenstehendes Beispiel.
- **fn** Funktion, die die Informationen über den Verarbeitungsfortschritt der Validierung erhält.
- **user_data** Benutzerdaten, die über den Callback an den Aufrufer zurückzusenden sind.
- **tableName** Optional. Eine bestimmte zu validierende Tabelle.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Je nach den an diese Routine übergebenen Parametern können Tabellen, Indizes und Datenbankseiten validiert werden. Um während der Validierung Informationen zu erhalten, implementieren Sie eine Callback-Funktion und übergeben die Adresse an diese Routine. Um die Validierung auf eine bestimmte Tabelle zu beschränken, übergeben Sie den Tabellennamen oder die ID als letzten Parameter.

Der flags-Parameter ist eine Kombination aus folgenden Werten:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Siehe auch

- ULVF_TABLE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 105
- ULVF_INDEX-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104
- ULVF_DATABASE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_EXPRESS-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_FULL_VALIDATE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104

Beispiel

```
The following example demonstrates table and index validation in express
mode:
flags = ULVF_TABLE | ULVF_INDEX | ULVF_EXPRESS;
```

ULDatabaseManager-Klasse

Verwaltet Verbindungen und Datenbanken.

Syntax

```
public class ULDatabaseManager
```

Mitglieder

Alle Mitglieder der ULDatabaseManager-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
CreateDatabase-Methode	Erstellt eine neue Datenbank.

Name	Beschreibung
DropDatabase-Methode	Löscht eine vorhandene Datenbank, die aktuell nicht läuft.
EnableAesDBEncryption-Methode	Aktiviert die AES Datenbankverschlüsselung.
EnableAesFipsDBEncryption-Methode	Aktiviert die FIPS 140-2-zertifizierte AES-Datenbankverschlüsselung.
EnableAllSynchronization-Methode	Aktiviert alle vier Synchronisationstypen: TCPIP, HTTP, TLS und HTTPS.
EnableHttpsSynchronization-Methode	Aktiviert die HTTPS-Synchronisation.
EnableHttpSynchronization-Methode	Aktiviert die HTTP-Synchronisation
EnableRsaE2ee-Methode	Aktiviert die RSA-Ende-zu-Ende-Verschlüsselung.
EnableRsaFipsE2ee-Methode	Aktiviert die FIPS 140-2-zertifizierte RSA-Ende-zu-Ende-Verschlüsselung.
EnableRsaFipsSyncEncryption-Methode	Aktiviert die FIPS 140-2-zertifizierte RSA-Synchronisationsverschlüsselung für SSL- oder TLS-Datenströme.
EnableRsaSyncEncryption-Methode	Aktiviert die RSA-Synchronisationsverschlüsselung.
EnableTcpipSynchronization-Methode	Aktiviert die TCP/IP-Synchronisation.
EnableTlsSynchronization-Methode	Aktiviert die TLS-Synchronisation.
EnableZlibSyncCompression-Methode	Aktiviert die Zlib-Komprimierung für einen Synchronisationsdatenstrom.
Fini-Methode	Finalisiert die UltraLite-Laufzeitumgebung.
Init-Methode	Initialisiert die UltraLite-Laufzeitumgebung.
OpenConnection-Methode	Öffnet eine neue Verbindung zu einer vorhandenen Datenbank.
SetErrorCallback-Methode	Legt den Callback fest, der bei einem Fehler aufzurufen ist.
ValidateDatabase-Methode	Führt eine Validierung der Basisdaten und eine Indexvalidierung in einer Datenbank durch.

Bemerkungen

Die Init-Methode muss in einer threadsicheren Umgebung aufgerufen werden, bevor andere Aufrufe ausgeführt werden können. Die Fini-Methode muss zum Abschluss in einer ähnlich threadsicheren Umgebung aufgerufen werden.

Hinweis
Diese Klasse ist statisch. Erstellen Sie davon keine Instanz.

CreateDatabase-Methode

Erstellt eine neue Datenbank.

Syntax

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

Parameter

- **connParms** Eine durch Semikola getrennte Zeichenfolge von Verbindungsparametern, die in der Form von Schlüsselwort=Wert-Paaren festgelegt werden. Die Verbindungszeichenfolge muss den Namen der Datenbank enthalten. Es handelt sich um dieselben Parameter, die beim Herstellen der Verbindung mit einer Datenbank angegeben werden können.
- **createParms** Eine durch Semikolons getrennte Zeichenfolge von Datenbankerstellungsparametern, die in der Form von Schlüsselwort-Wert-Paaren angegeben werden. Zum Beispiel: page_size=2048;obfuscate=yes.
- **error** Ein optionales ULError-Objekt für die Aufnahme der Fehlerinformationen.

Rückgabe

Ein ULConnection-Objekt für die neue Datenbank wird zurückgegeben, wenn die Datenbank erfolgreich erstellt wurde. NULL wird zurückgegeben, wenn die Methode fehlschlägt. Ein Fehler wird gewöhnlich durch einen ungültigen Dateinamen oder durch eine Verweigerung des Zugriffs verursacht.

Bemerkungen

Die Datenbank wird mit Informationen erstellt, die in zwei Parametergruppen bereitgestellt werden.

Der connParms-Parameter ist eine Gruppe von Standard-Verbindungsparametern, die immer anwendbar sind, wenn auf die Datenbank zugegriffen wird, z.B. Dateiname oder Chiffrierschlüssel.

Der createParms-Parameter ist eine Gruppe von Parametern, die nur zum Zeitpunkt der Datenbankerstellung relevant sind, z.B. Prüfsummenebene, Seitengröße, Kollation sowie Uhrzeit- und Datumsformat.

Der folgende Code zeigt, wie Sie die CreateDatabase-Methode verwenden, um eine UltraLite-Datenbank mit dem Dateinamen *mydb.udb* zu erstellen:

```
ULConnection * conn;
conn = ULDatabaseManager::CreateDatabase( "DBF=mydb.udb",
"checksum_level=2" );
if( conn != NULL ) {
    // success
} else {
    // unable to create
}
```

Siehe auch

- „UltraLite-Verbindungsparameter“ [*UltraLite - Datenbankverwaltung*]
- „UltraLite-Erstellungsparameter“ [*UltraLite - Datenbankverwaltung*]

DropDatabase-Methode

Löscht eine vorhandene Datenbank, die aktuell nicht läuft.

Syntax

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

Parameter

- **parms** Die Datenbank-Identifizierungsparameter. (Verbindungszeichenfolge)
- **error** Ein optionales ULError-Objekt für die Aufnahme der Fehlerinformationen.

Rückgabe

TRUE, wenn die Datenbank erfolgreich gelöscht wurde, ansonsten FALSE.

EnableAesDBEncryption-Methode

Aktiviert die AES Datenbankverschlüsselung.

Syntax

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

Bemerkungen

Rufen Sie diese Methode auf, um AES-Datenbankverschlüsselung zu verwenden. Verwenden Sie den DBKEY-Verbindungsparameter, um die Passphrase für die Verschlüsselung anzugeben. Sie müssen diese Methode aufrufen, bevor Sie die Datenbankverbindung öffnen.

Siehe auch

- „UltraLite-Verbindungsparameter DBKEY“ [*UltraLite - Datenbankverwaltung*]

EnableAesFipsDBEncryption-Methode

Aktiviert die FIPS 140-2-zertifizierte AES-Datenbankverschlüsselung.

Syntax

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

Bemerkungen

Rufen Sie diese Methode auf, um FIPS AES-Datenbankverschlüsselung zu verwenden. Verwenden Sie den DBKEY-Verbindungsparameter, um die Passphrase für die Verschlüsselung anzugeben.

Sie müssen in der Erstellungsparameter-Zeichenfolge der Datenbank "fips=yes" angeben. Sie müssen diese Methode aufrufen, bevor Sie die Datenbankverbindung öffnen.

Hinweis
Erforderliche getrennt lizenzierbare Komponenten.

FIPS-zertifizierte Verschlüsselung erfordert eine separate Lizenz. Alle Technologien für starke Verschlüsselungen unterliegen Exportbestimmungen.

Siehe „Getrennt lizenzierbare Komponenten“ [*SQL Anywhere 16 - Einführung*].

Siehe auch

- ULDatabaseManager.EnableAesDBEncryption-Methode [UltraLite C++] auf Seite 137
- „UltraLite-Verbindungsparameter DBKEY“ [*UltraLite - Datenbankverwaltung*]

EnableAllSynchronization-Methode

Aktiviert alle vier Synchronisationstypen: TCPIP, HTTP, TLS und HTTPS.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Setzen Sie beim Starten der Synchronisation den **stream**-Parameter auf TCPIP, HTTP, TLS oder HTTPS. Setzen Sie auch die Optionen für das Netzwerkprotokoll-Zertifikat, wenn Sie TLS oder HTTPS verwenden

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableHttpsSynchronization-Methode

Aktiviert die HTTPS-Synchronisation.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Setzen Sie beim Starten der Synchronisation den **stream**-Parameter auf "HTTPS". Legen Sie außerdem die Optionen für das Netzwerkprotokoll-Zertifikat fest.

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableHttpSynchronization-Methode

Aktiviert die HTTP-Synchronisation

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Setzen Sie beim Starten der Synchronisation den **stream**-Parameter auf "HTTP".

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableRsaE2ee-Methode

Aktiviert die RSA-Ende-zu-Ende-Verschlüsselung.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Um Ende-zu-Ende-Verschlüsselung zu verwenden, legen Sie die **e2ee_public_key**-Netzwerkprotokolloption fest.

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableRsaFipsE2ee-Methode

Aktiviert die FIPS 140-2-zertifizierte RSA-Ende-zu-Ende-Verschlüsselung.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Um Ende-zu-Ende-Verschlüsselung zu verwenden, legen Sie die **e2ee_public_key**-Netzwerkprotokolloption fest. In diesem Fall muss die **fips**-Option auf "yes" gesetzt werden.

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableRsaFipsSyncEncryption-Methode

Aktiviert die FIPS 140-2-zertifizierte RSA-Synchronisationsverschlüsselung für SSL- oder TLS-Datenströme.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Dies ist erforderlich, wenn Sie bei FIPS RSA-Verschlüsselung den **stream**-Parameter auf "TLS" oder "HTTPS" setzen. In diesem Fall muss die **fips**-Option auf "yes" gesetzt werden.

Siehe auch

- ULDatabaseManager.EnableRsaSyncEncryption-Methode [UltraLite C++] auf Seite 140
- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableRsaSyncEncryption-Methode

Aktiviert die RSA-Synchronisationsverschlüsselung.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Dies ist erforderlich, wenn Sie bei RSA-Verschlüsselung den **stream**-Parameter auf "TLS" oder "HTTPS" setzen.

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableTcpipSynchronization-Methode

Aktiviert die TCP/IP-Synchronisation.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Setzen Sie beim Starten der Synchronisation den **stream**-Parameter auf "TCPIP".

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableTlsSynchronization-Methode

Aktiviert die TLS-Synchronisation.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Setzen Sie beim Starten der Synchronisation den **stream**-Parameter auf "TLS". Legen Sie außerdem die Optionen für das Netzwerkprotokoll-Zertifikat fest.

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

EnableZlibSyncCompression-Methode

Aktiviert die Zlib-Komprimierung für einen Synchronisationsdatenstrom.

Syntax

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

Bemerkungen

Sie müssen diese Methode vor der Synchronize-Methode aufrufen.

Um Komprimierung zu verwenden, setzen Sie die **compression**-Netzwerkprotokolloption auf "zlib".

Siehe auch

- „Netzwerkprotokolloptionen des MobiLink-Clients“ [*MobiLink - Clientadministration*]

Fini-Methode

Finalisiert die UltraLite-Laufzeitumgebung.

Syntax

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

Bemerkungen

Diese Methode darf nur einmal von einem einzelnen Thread aufgerufen werden, wenn die Anwendung beendet ist. Sie ist nicht threadsicher.

Init-Methode

Initialisiert die UltraLite-Laufzeitumgebung.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE. FALSE kann auch zurückgegeben werden, wenn die Methode mehr als einmal aufgerufen wird.

Bemerkungen

Diese Methode darf nur einmal von einem einzelnen Thread aufgerufen werden, bevor andere Aufrufe ausgeführt werden können. Sie ist nicht threadsicher.

Diese Methode schlägt normalerweise nur fehl, wenn zu wenig Speicher verfügbar ist.

OpenConnection-Methode

Öffnet eine neue Verbindung zu einer vorhandenen Datenbank.

Syntax

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

Parameter

- **connParms** Die Verbindungszeichenfolge
- **error** Ein optionales ULError-Objekt für die Rückgabe der Fehlerinformationen.
- **reserved** Reserviert für interne Verwendung. Nicht angeben oder auf NULL setzen.

Rückgabe

Wenn die Methode erfolgreich ist, wird ein neues ULConnection-Objekt zurückgegeben, ansonsten NULL.

Bemerkungen

Die Verbindungszeichenfolge ist eine Gruppe von Option=Wert-Verbindungsparametern (durch Semikola getrennt), die angibt, mit welcher Datenbank verbunden werden soll und welche Optionen für die Verbindung verwendet werden sollen. Beispiel: Nachdem Sie Ihre Passphrase für die Verschlüsselung sicher abgerufen haben, lautet die resultierende Verbindungszeichenfolge möglicherweise "DBF=mydb.udb;DBKEY=iyntTZld9OEa#&G".

Um die Fehlerinformationen zu erhalten, übergeben Sie einen Zeiger auf ein ULError-Objekt. Nachstehend finden Sie eine Liste der möglichen Fehler:

- **SQLE_INVALID_PARSE_PARAMETER** **connParms** wurde nicht richtig formatiert.
- **SQLE_UNRECOGNIZED_OPTION** Ein Verbindungsoptionsname wurde wahrscheinlich falsch geschrieben.
- **SQLE_INVALID_OPTION_VALUE** Ein Verbindungsoptionswert wurde nicht richtig angegeben.
- **SQLE_ULTRALITE_DATABASE_NOT_FOUND** Die angegebene Datenbank wurde nicht gefunden.
- **SQLE_INVALID_LOGON** Sie haben eine ungültige Benutzer-ID oder ein falsches Kennwort eingegeben.
- **SQLE_TOO_MANY_CONNECTIONS** Die maximale Anzahl der parallelen Datenbankverbindungen wurde überschritten.

Siehe auch

- „UltraLite-Verbindungszeichenfolgen und Parameter“ [*UltraLite - Datenbankverwaltung*]
- „UltraLite-Verbindungsparameter“ [*UltraLite - Datenbankverwaltung*]

SetErrorCallback-Methode

Legt den Callback fest, der bei einem Fehler aufzurufen ist.

Syntax

```
public static void SetErrorCallback(
    ul_cpp_error_callback_fn callback,
    void * userData
)
```

Parameter

- **callback** Die Callback-Funktion.
- **userData** An den Callback übergebene Benutzerkontextinformationen.

Bemerkungen

Diese Methode ist nicht threadsicher.

ValidateDatabase-Methode

Führt eine Validierung der Basisdaten und eine Indexvalidierung in einer Datenbank durch.

Syntax

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

Parameter

- **connParms** Die Parameter für die Verbindung zur Datenbank.
- **flags** Die Parameter, die den Typ der Validierung steuern (siehe Beispiel unten).
- **fn** Funktion, die die Informationen über den Verarbeitungsfortschritt der Validierung erhält.
- **userData** Benutzerdaten, die über den Callback an den Aufrufer zurückzusenden sind.
- **error** Ein optionales ULError-Objekt für die Aufnahme der Fehlerinformationen.

Rückgabe

TRUE, wenn die Validierung erfolgreich ist, ansonsten FALSE.

Bemerkungen

Der flags-Parameter ist eine Kombination aus folgenden Werten:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Siehe auch

- ULVF_TABLE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 105
- ULVF_INDEX-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104
- ULVF_DATABASE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_EXPRESS-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_FULL_VALIDATE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104

Beispiel

Das folgende Beispiel zeigt die Tabelle und die Indexvalidierung im Express Modus:

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

ULDatabaseSchema-Klasse

Stellt das Schema einer UltraLite-Datenbank dar.

Syntax

```
public class ULDatabaseSchema
```

Mitglieder

Alle Mitglieder der ULDatabaseSchema-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Close-Methode	Zerstört dieses Objekt.
getConnection-Methode	Ruft das ULConnection-Objekt ab.
GetNextPublication-Methode	Ruft den Namen der nächsten Publikation in der Datenbank ab.
GetNextTable-Methode	Ruft die nächste Tabelle (das nächste Schema) in der Datenbank ab.
GetPublicationCount-Methode	Ruft die Anzahl der Publikationen in der Datenbank ab.
GetTableCount-Methode	Gibt die Anzahl der Tabellen in der Datenbank zurück.
GetTableSchema-Methode	Gibt das Schema der benannten Tabelle zurück.

Close-Methode

Zerstört dieses Objekt.

Syntax

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

getConnection-Methode

Ruft das ULConnection-Objekt ab.

Syntax

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

Rückgabe

Die mit diesem Objekt verbundene ULConnection.

GetNextPublication-Methode

Ruft den Namen der nächsten Publikation in der Datenbank ab.

Syntax

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

Parameter

- **iter** Ein Zeiger auf die Wiederholervariable.

Rückgabe

Der Name der nächsten Publikation. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten. NULL wird zurückgegeben, wenn die Wiederholung abgeschlossen ist.

Bemerkungen

Initialisieren Sie den iter-Wert vor dem ersten Aufruf auf die ul_publication_iter_start-Konstante.

Siehe auch

- ul_publication_iter_start-Variable [UltraLite C++] auf Seite 227

GetNextTable-Methode

Ruft die nächste Tabelle (das nächste Schema) in der Datenbank ab.

Syntax

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

Parameter

- **iter** Ein Zeiger auf die Wiederholervariable.

Rückgabe

Ein ULTableSchema-Objekt oder NULL, wenn die Wiederholung abgeschlossen ist.

Bemerkungen

Initialisieren Sie den iter-Wert vor dem ersten Aufruf auf die ul_table_iter_start-Konstante.

Siehe auch

- ul_table_iter_start-Variable [UltraLite C++] auf Seite 227

GetPublicationCount-Methode

Ruft die Anzahl der Publikationen in der Datenbank ab.

Syntax

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

Rückgabe

Die Anzahl von Publikationen in der Datenbank

Bemerkungen

Publikations-IDs reichen von 1 bis zu der von dieser Methode zurückgegebenen Nummer.

GetTableCount-Methode

Gibt die Anzahl der Tabellen in der Datenbank zurück.

Syntax

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

Rückgabe

Eine Ganzzahl, die die Anzahl der Tabellen angibt.

GetTableSchema-Methode

Gibt das Schema der benannten Tabelle zurück.

Syntax

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

Parameter

- **tableName** Der Name der Tabelle.

Rückgabe

Gibt ein ULTableSchema-Objekt für die angegebene Tabelle zurück und UL_NULL, wenn die Tabelle nicht vorhanden ist.

ULError-Klasse

Verwaltet die Fehler, die aus der UltraLite-Laufzeitumgebung zurückgegeben werden.

Syntax

```
public class ULError
```

Mitglieder

Alle Mitglieder der ULError-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
ULError-Konstruktor	Konstruiert ein ULError-Objekt.
Clear-Methode	Löscht den aktuellen Fehler.
GetErrorInfo-Methode	Gibt einen Zeiger auf das zugrunde liegende ul_error_info-Objekt zurück.
GetParameter-Methode	Kopiert die angegebenen Fehlerparameter in den bereitgestellten Puffer.
GetParameterCount-Methode	Gibt die Anzahl der Fehlerparameter zurück.
GetSQLCode-Methode	Gibt den SQLCODE-Fehlercode für den letzten Vorgang zurück.
GetSQLCount-Methode	Gibt einen Wert zurück, der vom letzten Vorgang abhängt, und das Ergebnis des Vorgangs.
GetString-Methode	Gibt die Beschreibung des aktuellen Fehlers zurück.
GetURL-Methode	Gibt eine URL zur Dokumentationsseite für diesen Fehler zurück.
IsOK-Methode	Testet den Fehlercode.

ULError-Konstruktor

Konstruiert ein ULError-Objekt.

Syntax

```
public ULError()
```

Clear-Methode

Löscht den aktuellen Fehler.

Syntax

```
public void Clear()
```

Bemerkungen

Der aktuelle Fehler wird bei den meisten Aufrufen automatisch gelöscht, daher wird dies von Anwendungen meist nicht aufgerufen.

GetErrorInfo-Methode

Gibt einen Zeiger auf das zugrunde liegende ul_error_info-Objekt zurück.

Überladungsliste

Name	Beschreibung
GetErrorInfo()-Methode	Gibt einen Zeiger auf das zugrunde liegende ul_error_info-Objekt zurück.
GetErrorInfo()-Methode	Gibt einen Zeiger auf das zugrunde liegende ul_error_info-Objekt zurück.

GetErrorInfo()-Methode

Gibt einen Zeiger auf das zugrunde liegende ul_error_info-Objekt zurück.

Syntax

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

Rückgabe

Ein Zeiger auf das zugrunde liegende ul_error_info-Objekt.

Siehe auch

- ul_error_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 95

GetErrorInfo()-Methode

Gibt einen Zeiger auf das zugrunde liegende ul_error_info-Objekt zurück.

Syntax

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

Rückgabe

Ein Zeiger auf das zugrunde liegende ul_error_info-Objekt.

Siehe auch

- ul_error_info-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 95

GetParameter-Methode

Kopiert die angegebenen Fehlerparameter in den bereitgestellten Puffer.

Syntax

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

Parameter

- **parmNo** Eine 1-basierte Parameternummer
- **dst** Der Puffer, der den Parameter aufnehmen soll
- **len** Die Größe des Puffers.

Rückgabe

Die für das Speichern des Parameters erforderliche Größe oder null, wenn die Ordinalzahl nicht gültig ist. Der Parameter wird gekürzt, wenn der Rückgabewert den len-Wert überschreitet.

Bemerkungen

Die Ausgabezeichenfolge wird immer mit NULL abgeschlossen, auch wenn der Puffer zu klein ist und die Zeichenfolge gekürzt wird.

GetParameterCount-Methode

Gibt die Anzahl der Fehlerparameter zurück.

Syntax

```
public ul_u_short GetParameterCount()
```

Rückgabe

Die Anzahl der Fehlerparameter.

GetSQLCode-Methode

Gibt den SQLCODE-Fehlercode für den letzten Vorgang zurück.

Syntax

```
public an_sql_code GetSQLCode()
```

Rückgabe

Der sqlcode-Wert.

GetSQLCount-Methode

Gibt einen Wert zurück, der vom letzten Vorgang abhängt, und das Ergebnis des Vorgangs.

Syntax

```
public ul_s_long GetSQLCount()
```

Rückgabe

Gibt den Wert für den letzten Vorgang zurück, falls zutreffend, ansonsten -1.

Bemerkungen

Die folgende Liste enthält die möglichen Vorgänge und die von diesen zurückgegebenen Ergebnisse:

- **INSERT-, UPDATE- oder DELETE-Vorgang erfolgreich ausgeführt** Gibt die Anzahl der Zeilen zurück, die von der Anweisung betroffen waren.
- **Syntaxfehler in SQL-Anweisung (SQLE_SYNTAX_ERROR)** Gibt die ungefähre Zeichenposition innerhalb der Anweisung zurück, an der der Fehler erkannt wurde.

GetString-Methode

Gibt die Beschreibung des aktuellen Fehlers zurück.

Syntax

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

Parameter

- **dst** Der Puffer, der die Fehlerbeschreibung aufnehmen soll
- **len** Die Größe des Puffers in Array-Elementen.

Rückgabe

Die erforderliche Größe zum Speichern der Zeichenfolge. Die Zeichenfolge wird gekürzt, wenn der Rückgabewert den len-Wert überschreitet.

Bemerkungen

Die Zeichenfolge enthält den Fehlercode und alle Parameter. Sie können eine vollständige Beschreibung des Fehlers abrufen, indem Sie die von der ULError.GetURL-Methode zurückgegebene URL laden.

Die Ausgabeparameter-Zeichenfolge ist immer mit einem Nullabschlusszeichen abgeschlossen, auch wenn der Puffer zu klein ist und die Zeichenfolge gekürzt wird.

Siehe auch

- ULError.GetURL-Methode [UltraLite C++] auf Seite 152

GetURL-Methode

Gibt eine URL zur Dokumentationsseite für diesen Fehler zurück.

Syntax

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

Parameter

- **buffer** Der Puffer, der die URL aufnehmen soll
- **len** Die Größe des Puffers.
- **reserved** Ist für eine spätere Verwendung reserviert. Sie müssen NULL übergeben, die Standardeinstellung.

Rückgabe

Die erforderliche Größe zum Speichern der URL. Die URL wird gekürzt, wenn der Rückgabewert den len-Wert überschreitet.

IsOK-Methode

Testet den Fehlercode.

Syntax

```
public bool IsOK()
```

Rückgabe

TRUE, wenn der aktuelle Code SQLE_NOERROR oder eine Warnung ist, und FALSE, wenn der aktuelle Code einen Fehler anzeigt.

ULIndexSchema-Klasse

Stellt das Schema eines UltraLite-Tabellenindexes dar.

Syntax

```
public class ULIndexSchema
```

Mitglieder

Alle Mitglieder der ULIndexSchema-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Close-Methode	Zerstört dieses Objekt.
GetColumnCount-Methode	Ruft die Anzahl der Spalten im Index ab.

Name	Beschreibung
GetColumnName-Methode	Ruft den Namen der Spalte anhand der Position der Spalte im Index ab.
getConnection-Methode	Ruft das ULConnection-Objekt ab.
GetIndexColumnID-Methode	Ruft die Spalten-ID des 1-basierten Indexes über den Namen ab.
GetIndexFlags-Methode	Ruft das Bitfeld des Indexeigenschaft-Flag-Parameters ab.
GetName-Methode	Ruft den Namen des Indexes ab.
GetReferencedIndexName-Methode	Ruft den zugeordneten Primärindexnamen ab.
GetReferencedTableName-Methode	Ruft den zugeordneten Primärtabellennamen ab.
GetTableName-Methode	Ruft den Namen der Tabelle ab, die den Index enthält.
IsColumnDescending-Methode	Ermittelt, ob die Spalte absteigend sortiert ist.

Close-Methode

Zerstört dieses Objekt.

Syntax

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

GetColumnCount-Methode

Ruft die Anzahl der Spalten im Index ab.

Syntax

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

Rückgabe

Die Anzahl der Spalten im Index

GetColumnName-Methode

Ruft den Namen der Spalte anhand der Position der Spalte im Index ab.

Syntax

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

Parameter

- **col_id_in_index** Die 1-basierte Ordinalnummer, die die Position der Spalte im Index anzeigt

Rückgabe

Der Name der Spalte. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

getConnection-Methode

Ruft das ULConnection-Objekt ab.

Syntax

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

Rückgabe

Die diesem Objekt zugeordnete Verbindung.

GetIndexColumnID-Methode

Ruft die Spalten-ID des 1-basierten Indexes über den Namen ab.

Syntax

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

Parameter

- **columnName** Der Spaltenname.

Rückgabe

Gibt 0 zurück und setzt SQLE_COLUMN_NOT_FOUND, wenn der Spaltenname nicht vorhanden ist.

GetIndexFlags-Methode

Ruft das Bitfeld des Indexeigenschaft-Flag-Parameters ab.

Syntax

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

Siehe auch

- ul_index_flag-Enumeration [UltraLite C++] auf Seite 225

GetName-Methode

Ruft den Namen des Indexes ab.

Syntax

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

Rückgabe

Der Name des Indexes. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

GetReferencedIndexName-Methode

Ruft den zugeordneten Primärindexnamen ab.

Syntax

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

Rückgabe

Der Name des referenzierten Indexes. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

Bemerkungen

Diese Methode gilt nur für Fremdschlüssel.

GetReferencedTableName-Methode

Ruft den zugeordneten Primärtabellennamen ab.

Syntax

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

Rückgabe

Der Name der referenzierten Tabelle. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

Bemerkungen

Diese Methode gilt nur für Fremdschlüssel.

GetTableName-Methode

Ruft den Namen der Tabelle ab, die den Index enthält.

Syntax

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

Rückgabe

Der Name der Tabelle, die diese Index enthält. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

IsColumnDescending-Methode

Ermittelt, ob die Spalte absteigend sortiert ist.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

TRUE, wenn die Spalte in absteigender Reihenfolge sortiert ist, ansonsten FALSE.

ULPreparedStatement-Klasse

Stellt eine vorbereitete SQL-Anweisung dar.

Syntax

```
public class ULPreparedStatement
```

Mitglieder

Alle Mitglieder der ULPreparedStatement-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
AppendParameterByteChunk-Methode	Legt einen großen Binär-Parameter, aufgeteilt in mehrere Abschnitte, fest.
AppendParameterStringChunk-Methode	Legt einen großen Zeichenfolgen-Parameter, aufgeteilt in mehrere Abschnitte fest.
Close-Methode	Zerstört dieses Objekt.

Name	Beschreibung
ExecuteQuery-Methode	Führt eine SQL SELECT-Anweisung als Abfrage aus.
ExecuteStatement-Methode	Führt eine Anweisung aus, die keine Ergebnismenge, wie etwa eine INSERT- oder UPDATE-Anweisung, zurückgibt.
getConnection-Methode	Ruft das Verbindungsobjekt ab.
GetParameterCount-Methode	Ruft die Anzahl der Eingabeparameter für diese Anweisung ab.
GetParameterID-Methode	Ruft die 1-basierte Ordinalzahl für einen Parameternamen auf.
GetParameterType-Methode	Ruft den Speicher-/Hostvariablentyp eines Parameters ab.
GetPlan-Methode	Ruft eine Beschreibung des Abfrageausführungsplans in Textform ab.
GetResultSetSchema-Methode	Ruft das Schema für die Ergebnismenge ab.
GetRowsAffectedCount-Methode	Ruft die Anzahl der Zeilen ab, die von der letzten SQL-Anweisung betroffen sind.
HasResultSet-Methode	Ermittelt, ob die SQL-Anweisung eine Ergebnismenge hat.
SetParameterBinary-Methode	Legt einen Parameter mit einem ul_binary-Wert fest.
SetParameterDateTime-Methode	Legt einen Parameter mit einem DECL_DATETIME-Wert fest.
SetParameterDouble-Methode	Legt einen Parameter mit einem double-Wert fest.
SetParameterFloat-Methode	Legt einen Parameter mit einem float-Wert fest.
SetParameterGuid-Methode	Legt einen Parameter mit einem GUID-Wert fest.
SetParameterInt-Methode	Legt einen Parameter mit einem Ganzzahlwert fest.
SetParameterIntWithType-Methode	Legt einen Parameter mit einem Ganzzahlwert des angegebenen integer-Typs fest.
SetParameterNull-Methode	Legt einen Parameter mit NULL fest.
SetParameterString-Methode	Legt einen Parameter mit einem Zeichenfolgenwert fest.

AppendParameterByteChunk-Methode

Legt einen großen Binär-Parameter, aufgeteilt in mehrere Abschnitte, fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der Byte Abschnitt, der angehängt werden soll.
- **valueSize** Die Größe des Puffers.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

AppendParameterStringChunk-Methode

Legt einen großen Zeichenfolgen-Parameter, aufgeteilt in mehrere Abschnitte fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der Zeichenfolgenabschnitt, der angehängt werden soll.
- **len** Optional. Setzen Sie dies auf die Länge des Zeichenfolgenabschnitts in Byte oder auf UL_NULL_TERMINATED_STRING, wenn der Zeichenfolgenabschnitt mit Null abgeschlossen ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Close-Methode

Zerstört dieses Objekt.

Syntax

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

ExecuteQuery-Methode

Führt eine SQL SELECT-Anweisung als Abfrage aus.

Syntax

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

Rückgabe

Das ULResultSet-Objekt, das die Ergebnisse der Abfrage als Zeilenmenge enthält.

ExecuteStatement-Methode

Führt eine Anweisung aus, die keine Ergebnismenge, wie etwa eine INSERT- oder UPDATE-Anweisung, zurückgibt.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

getConnection-Methode

Ruft das Verbindungsobjekt ab.

Syntax

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

Rückgabe

Das dieser vorbereiteten Anweisung zugeordnete ULConnection-Objekt.

GetParameterCount-Methode

Ruft die Anzahl der Eingabeparameter für diese Anweisung ab.

Syntax

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

Rückgabe

Die Anzahl der Eingabeparameter für diese Anweisung.

GetParameterID-Methode

Ruft die 1-basierte Ordinalzahl für einen Parameternamen auf.

Syntax

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

Parameter

- **name** Der Name der Hostvariablen.

Rückgabe

Die 1-basierte Ordinalzahl für einen Parameternamen.

GetParameterType-Methode

Ruft den Speicher-/Hostvariablentyp eines Parameters ab.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters

Rückgabe

Der Typ des angegeben Parameters.

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

GetPlan-Methode

Ruft eine Beschreibung des Abfrageausführungsplans in Textform ab.

Syntax

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

Parameter

- **dst** Der Zielpuffer für den Plantext. Übergeben Sie NULL, um die Größe des erforderlichen Puffers für den Plan festzulegen.
- **dstSize** Die Größe des Zielpuffers.

Rückgabe

Die Anzahl von Byte, die in den Puffer kopiert wurden. Andernfalls, wenn der dst-Wert NULL ist, wird die Anzahl der zum Speichern des Plans erforderlichen Byte (ohne das Nullabschlusszeichen) zurückgegeben.

Bemerkungen

Diese Methode ist in erster Linie für die Entwicklung gedacht.

Eine leere Zeichenfolge wird zurückgegeben, wenn es keinen Plan gibt. Pläne sind vorhanden, wenn die vorbereitete Anweisung eine SQL-Abfrage ist.

Wenn der Plan bezogen wird, bevor die damit zusammenhängende Abfrage ausgeführt wurde, zeigt der Plan die Vorgänge, die zum Ausführen der Abfrage verwendet werden. Der Plan zeigt darüber hinaus die Anzahl von Zeilen, die jeder Vorgang produziert hat, wenn der Plan bezogen wird, nachdem die Abfrage ausgeführt wurde. Dieser Plan kann verwendet werden, um Einblicke in die Ausführung der Abfrage zu erhalten.

GetResultSetSchema-Methode

Ruft das Schema für die Ergebnismenge ab.

Syntax

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

Rückgabe

Ein ULResultSetSchema-Objekt, das verwendet werden kann, um Informationen über das Schema der Ergebnismenge zu beziehen.

GetRowsAffectedCount-Methode

Ruft die Anzahl der Zeilen ab, die von der letzten SQL-Anweisung betroffen sind.

Syntax

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

Rückgabe

Anzahl der Zeilen, die von der letzten Anweisung betroffen waren. Wenn die Anzahl der Zeilen nicht verfügbar ist (zum Beispiel wenn die Anweisung das Schema und nicht die Daten ändert), ist der Rückgabewert -1.

HasResultSet-Methode

Ermittelt, ob die SQL-Anweisung eine Ergebnismenge hat.

Syntax

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

Rückgabe

TRUE, wenn beim Ausführen dieser Anweisung eine Ergebnismenge generiert wird, andernfalls FALSE.

SetParameterBinary-Methode

Legt einen Parameter mit einem ul_binary-Wert fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der ul_binary-Wert.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterDateTime-Methode

Legt einen Parameter mit einem DECL_DATETIME-Wert fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der DECL_DATETIME-Wert.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterDouble-Methode

Legt einen Parameter mit einem double-Wert fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der double-Wert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterFloat-Methode

Legt einen Parameter mit einem float-Wert fest.

Syntax

```
public virtual bool SetParameterFloat(ul_column_num pid, ul_real value)
```

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der float-Wert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterGuid-Methode

Legt einen Parameter mit einem GUID-Wert fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der GUID-Wert.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterInt-Methode

Legt einen Parameter mit einem Ganzzahlwert fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der Ganzzahlwert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterIntWithType-Methode

Legt einen Parameter mit einem Ganzzahlwert des angegebenen integer-Typs fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der Ganzzahlwert
- **type** Der integer-Datentyp, mit dem der Wert zu behandeln ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Nachstehend finden Sie eine Liste der Ganzzahlwerte, die für den value-Parameter verwendet werden können:

- 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

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

SetParameterNull-Methode

Legt einen Parameter mit NULL fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetParameterString-Methode

Legt einen Parameter mit einem Zeichenfolgenwert fest.

Syntax

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

Parameter

- **pid** Die 1-basierte Ordinalzahl des Parameters
- **value** Der Zeichenfolgenwert.
- **len** Optional. Setzen Sie dies auf die Länge der Zeichenfolge in Byte oder auf UL_NULL_TERMINATED_STRING, wenn die Zeichenfolge mit Null abgeschlossen ist. SQLE_INVALID_PARAMETER wird gesetzt, wenn dieser Parameter größer ist als 32 kB. Rufen Sie bei großen Zeichenfolgen stattdessen die AppendParameterStringChunk-Methode auf.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Siehe auch

- ULPreparedStatement.AppendParameterStringChunk-Methode [UltraLite C++] auf Seite 158

ULResultSet-Klasse

Stellt eine Ergebnismenge in einer UltraLite-Datenbank dar.

Syntax

```
public class ULResultSet
```

Abgeleitete Klassen

- ULTable-Klasse [UltraLite C++] auf Seite 207

Mitglieder

Alle Mitglieder der ULResultSet-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
AfterLast-Methode	Verschiebt den Cursor hinter die letzte Zeile.
AppendByteChunk-Methode	Hängt Byte an eine Spalte an.
AppendStringChunk-Methode	Hängt einen Zeichenfolgenabschnitt an eine Spalte an.
BeforeFirst-Methode	Verschiebt den Cursor vor die erste Zeile.
Close-Methode	Zerstört dieses Objekt.
Delete-Methode	Löscht die aktuelle Zeile und verschiebt den Cursor auf die nächste gültige Zeile.
DeleteNamed-Methode	Löscht die aktuelle Zeile und verschiebt den Cursor auf die nächste gültige Zeile.
First-Methode	Verschiebt den Cursor in die erste Zeile.
GetBinary-Methode	Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.
GetBinaryLength-Methode	Ruft die binäre Länge des Werts einer Spalte ab.
GetByteChunk-Methode	Ruft den binären Abschnitt aus der Spalte ab.
getConnection-Methode	Ruft das Verbindungsobjekt ab.
GetDateTime-Methode	Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.
GetDouble-Methode	Ruft einen Wert aus einer Spalte als double-Typ ab.
GetFloat-Methode	Ruft einen Wert aus einer Spalte als float-Typ ab.
GetGuid-Methode	Ruft einen Wert aus einer Spalte als GUID ab.

Name	Beschreibung
GetInt-Methode	Ruft einen Wert aus einer Spalte als integer-Typ ab.
GetIntWithType-Methode	Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.
GetResultSetSchema-Methode	Gibt ein Objekt zurück, das verwendet werden kann, um Informationen über die Ergebnismenge zu beziehen.
GetRowCount-Methode	Ruft die Anzahl der Zeilen in der Tabelle ab.
GetState-Methode	Ruft den internen Zustand des Cursors ab.
GetString-Methode	Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.
GetStringChunk-Methode	Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.
GetStringLength-Methode	Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.
IsNull-Methode	Überprüft, ob eine Spalte NULL ist.
Last-Methode	Verschiebt den Cursor in die letzte Zeile.
Next-Methode	Verschiebt den Cursor eine Zeile vorwärts.
Previous-Methode	Verschiebt den Cursor eine Zeile zurück.
Relative-Methode	Verschiebt den Cursor von der aktuellen Cursorposition ausgehend um die von Offset angegebene Anzahl von Zeilen.
SetBinary-Methode	Legt eine Spalte mit einem ul_binary-Wert fest.
SetDateTime-Methode	Legt eine Spalte mit einem DECL_DATETIME-Wert fest.
SetDefault-Methode	Setzt eine Spalte auf ihren Standardwert.
SetDouble-Methode	Legt eine Spalte mit einem double-Wert fest.
SetFloat-Methode	Legt eine Spalte mit einem float-Wert fest.
SetGuid-Methode	Legt eine Spalte mit einem GUID-Wert fest.
SetInt-Methode	Legt eine Spalte mit einem Ganzzahlwert fest.
SetIntWithType-Methode	Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.
SetNull-Methode	Setzt eine Spalte auf NULL.

Name	Beschreibung
SetString-Methode	Legt eine Spalte mit einem Zeichenfolgenwert fest.
Update-Methode	Aktualisiert die aktuelle Zeile.
UpdateBegin-Methode	Wählt den Aktualisierungsmodus zum Festlegen von Spalten aus.

AfterLast-Methode

Verschiebt den Cursor hinter die letzte Zeile.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

AppendByteChunk-Methode

Hängt Byte an eine Spalte an.

Überladungsliste

Name	Beschreibung
AppendByteChunk(const char *, const ul_byte *, size_t)-Methode	Hängt Byte an eine Spalte an.
AppendByteChunk(ul_column_num, const ul_byte *, size_t)-Methode	Hängt Byte an eine Spalte an.

AppendByteChunk(const char *, const ul_byte *, size_t)-Methode

Hängt Byte an eine Spalte an.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der Byte Abschnitt, der angehängt werden soll.

- **valueSize** Die Größe des Byte-Abschnitts in Byte.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Die angegebenen Byte werden am Ende der Spalte angefügt, in die bisher von AppendBinaryChunk-Methodenaufrufen geschrieben wurde.

Siehe auch

- ULResultSet.AppendByteChunk-Methode [UltraLite C++] auf Seite 168

AppendByteChunk(ul_column_num, const ul_byte *, size_t)-Methode

Hängt Byte an eine Spalte an.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der Byte Abschnitt, der angehängt werden soll.
- **valueSize** Die Größe des Byte-Abschnitts in Byte.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Die angegebenen Byte werden am Ende der Spalte angefügt, in die bisher von AppendBinaryChunk-Methodenaufrufen geschrieben wurde.

Siehe auch

- ULResultSet.AppendByteChunk-Methode [UltraLite C++] auf Seite 168

AppendStringChunk-Methode

Hängt einen Zeichenfolgenabschnitt an eine Spalte an.

Überladungsliste

Name	Beschreibung
AppendStringChunk(const char *, const char *, size_t)-Methode	Hängt einen Zeichenfolgenabschnitt an eine Spalte an.
AppendStringChunk(ul_column_num, const char *, size_t)-Methode	Hängt einen Zeichenfolgenabschnitt an eine Spalte an.

AppendStringChunk(const char *, const char *, size_t)-Methode

Hängt einen Zeichenfolgenabschnitt an eine Spalte an.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der Zeichenfolgenabschnitt, der angehängt werden soll.
- **len** Optional. Die Länge des Zeichenfolgenabschnitts in Byte oder die UL_NULL_TERMINATED_STRING-Konstante, wenn der Zeichenfolgenabschnitt mit Null abgeschlossen ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Diese Methode hängt die angegebene Zeichenfolge am Ende der Zeichenfolge an, die bisher von AppendStringChunk-Methodenaufrufen geschrieben wurde.

Siehe auch

- ULResultSet.AppendStringChunk-Methode [UltraLite C++] auf Seite 169

AppendStringChunk(ul_column_num, const char *, size_t)-Methode

Hängt einen Zeichenfolgenabschnitt an eine Spalte an.

Syntax

```
public virtual bool AppendStringChunk(
    ul_column_num cid,
```

```
    const char * value,
    size_t len
)
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der Zeichenfolgenabschnitt, der angehängt werden soll.
- **len** Optional. Die Länge des Zeichenfolgenabschnitts in Byte oder die UL_NULL_TERMINATED_STRING-Konstante, wenn der Zeichenfolgenabschnitt mit Null abgeschlossen ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Diese Methode hängt die angegebene Zeichenfolge am Ende der Zeichenfolge an, die bisher von AppendStringChunk-Methodenaufrufen geschrieben wurde.

Siehe auch

- ULResultSet.AppendStringChunk-Methode [UltraLite C++] auf Seite 169

BeforeFirst-Methode

Verschiebt den Cursor vor die erste Zeile.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Close-Methode

Zerstört dieses Objekt.

Syntax

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

Delete-Methode

Löscht die aktuelle Zeile und verschiebt den Cursor auf die nächste gültige Zeile.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

DeleteNamed-Methode

Löscht die aktuelle Zeile und verschiebt den Cursor auf die nächste gültige Zeile.

Syntax

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

Parameter

- **tableName** Ein Tabellenname oder seine Korrelation (erforderlich, wenn die Datenbank mehrere Spalten mit identischem Tabellennamen hat).

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

First-Methode

Verschiebt den Cursor in die erste Zeile.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

GetBinary-Methode

Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.

Überladungsliste

Name	Beschreibung
GetBinary(const char *, p_ul_binary, size_t)-Methode	Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.
GetBinary(ul_column_num, p_ul_binary, size_t)-Methode	Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.

GetBinary(const char *, p_ul_binary, size_t)-Methode

Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **dst** Das ul_binary-Ergebnis.
- **len** Die Größe des ul_binary-Objekts.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

GetBinary(ul_column_num, p_ul_binary, size_t)-Methode

Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **dst** Das ul_binary-Ergebnis.
- **len** Die Größe des ul_binary-Objekts.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

GetBinaryLength-Methode

Ruft die binäre Länge des Werts einer Spalte ab.

Überladungsliste

Name	Beschreibung
GetBinaryLength(const char *)-Methode	Ruft die binäre Länge des Werts einer Spalte ab.
GetBinaryLength(ul_column_num)-Methode	Ruft die binäre Länge des Werts einer Spalte ab.

GetBinaryLength(const char *)-Methode

Ruft die binäre Länge des Werts einer Spalte ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

Die Größe des Spaltenwerts als binary-Typ

GetBinaryLength(ul_column_num)-Methode

Ruft die binäre Länge des Werts einer Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Die Größe des Spaltenwerts als binary-Typ

GetByteChunk-Methode

Ruft den binären Abschnitt aus der Spalte ab.

Überladungsliste

Name	Beschreibung
GetByteChunk(const char *, ul_byte *, size_t, size_t)-Methode	Ruft den binären Abschnitt aus der Spalte ab.

Name	Beschreibung
GetByteChunk(ul_column_num, ul_byte *, size_t, size_t)-Methode	Ruft den binären Abschnitt aus der Spalte ab.

GetByteChunk(const char *, ul_byte *, size_t, size_t)-Methode

Ruft den binären Abschnitt aus der Spalte ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **dst** Der Puffer zur Aufnahme der Byte.
- **len** Die Größe des Puffers in Byte.
- **offset** Der Offset zu dem Wert, an dem der Lesevorgang beginnen soll, oder die UL_BLOB_CONTINUE-Konstante, um dort fortzufahren, wo der letzte Lesevorgang geendet hat.

Rückgabe

Die Anzahl der Byte, die in den Zielpuffer kopiert wurden. Wenn der dst-Wert NULL ist, wird die Anzahl der verbleibenden Byte zurückgegeben. Im dst-Parameter wird eine leere Zeichenfolge zurückgegeben, wenn die Spalte Null ist. Verwenden Sie die IsNull-Methode, um zwischen Null und leeren Zeichenfolgen zu unterscheiden.

Bemerkungen

Das Ende des Werts wurde erreicht, wenn 0 zurückgegeben wird.

Siehe auch

- ULResultSet.IsNull-Methode [UltraLite C++] auf Seite 189

GetByteChunk(ul_column_num, ul_byte *, size_t, size_t)-Methode

Ruft den binären Abschnitt aus der Spalte ab.

Syntax

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

```
        size_t len,
        size_t offset
    )
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **dst** Der Puffer zur Aufnahme der Byte.
- **len** Die Größe des Puffers in Byte.
- **offset** Der Offset zu dem Wert, an dem der Lesevorgang beginnen soll, oder die UL_BLOB_CONTINUE-Konstante, um dort fortzufahren, wo der letzte Lesevorgang geendet hat.

Rückgabe

Die Anzahl der Byte, die in den Zielpuffer kopiert wurden. Wenn der dst-Wert NULL ist, wird die Anzahl der verbleibenden Byte zurückgegeben. Im dst-Parameter wird eine leere Zeichenfolge zurückgegeben, wenn die Spalte Null ist. Verwenden Sie die IsNull-Methode, um zwischen Null und leeren Zeichenfolgen zu unterscheiden.

Bemerkungen

Das Ende des Werts wurde erreicht, wenn 0 zurückgegeben wird.

Siehe auch

- ULResultSet.IsNull-Methode [UltraLite C++] auf Seite 189

getConnection-Methode

Ruft das Verbindungsobjekt ab.

Syntax

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

Rückgabe

Das dieser Ergebnismenge zugeordnete ULConnection-Objekt.

GetDateTime-Methode

Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.

Überladungsliste

Name	Beschreibung
GetDateTime(const char *, DECL_DATETIME *)-Methode	Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.

Name	Beschreibung
GetDateTime(ul_column_num, DECL_DATETIME *)-Methode	Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.

GetDateTime(const char *, DECL_DATETIME *)-Methode

Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **dst** Der DECL_DATETIME-Wert.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

GetDateTime(ul_column_num, DECL_DATETIME *)-Methode

Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **dst** Der DECL_DATETIME-Wert.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

GetDouble-Methode

Ruft einen Wert aus einer Spalte als double-Typ ab.

Überladungsliste

Name	Beschreibung
GetDouble(const char *)-Methode	Ruft einen Wert aus einer Spalte als double-Typ ab.

Name	Beschreibung
GetDouble(ul_column_num)-Methode	Ruft einen Wert aus einer Spalte als double-Typ ab.

GetDouble(const char *)-Methode

Ruft einen Wert aus einer Spalte als double-Typ ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

Der Spaltenwert als double-Datentyp.

GetDouble(ul_column_num)-Methode

Ruft einen Wert aus einer Spalte als double-Typ ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Der Spaltenwert als double-Datentyp.

GetFloat-Methode

Ruft einen Wert aus einer Spalte als float-Typ ab.

Überladungsliste

Name	Beschreibung
GetFloat(const char *)-Methode	Ruft einen Wert aus einer Spalte als float-Typ ab.
GetFloat(ul_column_num)-Methode	Ruft einen Wert aus einer Spalte als float-Typ ab.

GetFloat(const char *)-Methode

Ruft einen Wert aus einer Spalte als float-Typ ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

Der Spaltenwert als float-Datentyp

GetFloat(ul_column_num)-Methode

Ruft einen Wert aus einer Spalte als float-Typ ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Der Spaltenwert als float-Datentyp

GetGuid-Methode

Ruft einen Wert aus einer Spalte als GUID ab.

Überladungsliste

Name	Beschreibung
GetGuid(const char *, GUID *)-Methode	Ruft einen Wert aus einer Spalte als GUID ab.
GetGuid(ul_column_num, GUID *)-Methode	Ruft einen Wert aus einer Spalte als GUID ab.

GetGuid(const char *, GUID *)-Methode

Ruft einen Wert aus einer Spalte als GUID ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **dst** Der GUID-Wert.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

GetGuid(ul_column_num, GUID *)-Methode

Ruft einen Wert aus einer Spalte als GUID ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **dst** Der GUID-Wert.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

GetInt-Methode

Ruft einen Wert aus einer Spalte als integer-Typ ab.

Überladungsliste

Name	Beschreibung
GetInt(const char *)-Methode	Ruft einen Wert aus einer Spalte als integer-Typ ab.
GetInt(ul_column_num)-Methode	Ruft einen Wert aus einer Spalte als integer-Typ ab.

GetInt(const char *)-Methode

Ruft einen Wert aus einer Spalte als integer-Typ ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

Der Spaltenwert als integer-Datentyp.

GetInt(ul_column_num)-Methode

Ruft einen Wert aus einer Spalte als integer-Typ ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Der Spaltenwert als integer-Datentyp.

GetIntWithType-Methode

Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.

Überladungsliste

Name	Beschreibung
GetIntWithType(const char *, ul_column_storage_type)-Methode	Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.
GetIntWithType(ul_column_num, ul_column_storage_type)-Methode	Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.

GetIntWithType(const char *, ul_column_storage_type)-Methode

Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **type** Der integer-Datentyp für den Aufruf.

Rückgabe

Der Spaltenwert als integer-Datentyp.

Bemerkungen

Nachstehend finden Sie eine Liste der Ganzzahlwerte, die für den type-Parameter verwendet werden können:

- 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

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

GetIntWithType(ul_column_num, ul_column_storage_type)-Methode

Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **type** Der integer-Datentyp für den Aufruf.

Rückgabe

Der Spaltenwert als integer-Datentyp.

Bemerkungen

Nachstehend finden Sie eine Liste der Ganzzahlwerte, die für den type-Parameter verwendet werden können:

- 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

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

GetResultSetSchema-Methode

Gibt ein Objekt zurück, das verwendet werden kann, um Informationen über die Ergebnismenge zu beziehen.

Syntax

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

Rückgabe

Ein ULResultSetSchema-Objekt, das verwendet werden kann, um Informationen über die Ergebnismenge zu beziehen.

GetRowCount-Methode

Ruft die Anzahl der Zeilen in der Tabelle ab.

Syntax

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

Parameter

- **threshold** Das Limit für die Anzahl der zu zählenden Zeilen. Setzen Sie dies auf 0, um anzugeben, dass keine Beschränkung festgelegt wird.

Rückgabe

Die Anzahl der Zeilen in der Tabelle.

Bemerkungen

Diese Methode entspricht dem Ausführen der Anweisung "SELECT COUNT(*) FROM table".

GetState-Methode

Ruft den internen Zustand des Cursors ab.

Syntax

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

Rückgabe

Der Status des Cursors.

Siehe auch

- UL_RS_STATE-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 88

GetString-Methode

Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.

Überladungsliste

Name	Beschreibung
GetString(const char *, char *, size_t)-Methode	Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.
GetString(ul_column_num, char *, size_t)-Methode	Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.

GetString(const char *, char *, size_t)-Methode

Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **dst** Der Puffer zur Aufnahme des Zeichenfolgenwerts. Die Zeichenfolge wird mit einem Nullwert abgeschlossen, auch wenn sie gekürzt ist.
- **len** Die Größe des Puffers in Byte.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

Bemerkungen

Die Zeichenfolge wird im Puffer gekürzt, wenn dieser nicht groß genug ist, um den gesamten Wert aufzunehmen.

GetString(ul_column_num, char *, size_t)-Methode

Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **dst** Der Puffer zur Aufnahme des Zeichenfolgenwerts. Die Zeichenfolge wird mit einem Nullwert abgeschlossen, auch wenn sie gekürzt ist.
- **len** Die Größe des Puffers in Byte.

Rückgabe

TRUE, wenn der Wert erfolgreich abgerufen wurde.

Bemerkungen

Die Zeichenfolge wird im Puffer gekürzt, wenn dieser nicht groß genug ist, um den gesamten Wert aufzunehmen.

GetStringChunk-Methode

Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.

Überladungsliste

Name	Beschreibung
GetStringChunk(const char *, char *, size_t, size_t)-Methode	Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.
GetStringChunk(ul_column_num, char *, size_t, size_t)-Methode	Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.

GetStringChunk(const char *, char *, size_t, size_t)-Methode

Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **dst** Der Puffer zur Aufnahme des Zeichenfolgenabschnitts. Die Zeichenfolge wird mit einem Nullwert abgeschlossen, auch wenn sie gekürzt ist.
- **len** Die Größe des Puffers in Byte.
- **offset** Der Offset zu dem Wert, an dem der Lesevorgang beginnen soll, oder die UL_BLOB_CONTINUE-Konstante, um dort fortzufahren, wo der letzte Lesevorgang geendet hat.

Rückgabe

Die Anzahl der in den Zielpuffer kopierten Byte, ausgenommen das Nullabschlusszeichen. Wenn der dst-Wert auf NULL gesetzt ist, wird die Anzahl der verbleibenden Byte in der Zeichenfolge zurückgegeben. Im dst-Parameter wird eine leere Zeichenfolge zurückgegeben, wenn die Spalte Null ist. Verwenden Sie die IsNull-Methode, um zwischen Null und leeren Zeichenfolgen zu unterscheiden.

Bemerkungen

Das Ende des Werts wurde erreicht, wenn 0 zurückgegeben wird.

Siehe auch

- ULResultSet.IsNull-Methode [UltraLite C++] auf Seite 189

GetStringChunk(ul_column_num, char *, size_t, size_t)-Methode

Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **dst** Der Puffer zur Aufnahme des Zeichenfolgenabschnitts. Die Zeichenfolge wird mit einem Nullwert abgeschlossen, auch wenn sie gekürzt ist.
- **len** Die Größe des Puffers in Byte.
- **offset** Setzen Sie dies auf den Offset zu dem Wert, an dem der Lesevorgang beginnen soll, oder auf die UL_BLOB_CONTINUE-Konstante, um dort fortzufahren, wo der letzte Lesevorgang geendet hat.

Rückgabe

Die Anzahl der in den Zielpuffer kopierten Byte, ausgenommen das Nullabschlusszeichen. Wenn der dst-Wert auf NULL gesetzt ist, wird die Anzahl der verbleibenden Byte in der Zeichenfolge zurückgegeben. Im dst-Parameter wird eine leere Zeichenfolge zurückgegeben, wenn die Spalte Null ist. Verwenden Sie die IsNull-Methode, um zwischen Null und leeren Zeichenfolgen zu unterscheiden.

Bemerkungen

Das Ende des Werts wurde erreicht, wenn 0 zurückgegeben wird.

Siehe auch

- ULResultSet.IsNull-Methode [UltraLite C++] auf Seite 189

GetStringLength-Methode

Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.

Überladungsliste

Name	Beschreibung
GetStringLength(const char *)-Methode	Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.
GetStringLength(ul_column_num)-Methode	Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.

GetStringLength(const char *)-Methode

Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

Die Anzahl von Byte oder Zeichen (ohne Nullabschlusszeichen), die erforderlich sind, um die von einer der GetString-Methoden zurückgegebene Zeichenfolge zu speichern.

Bemerkungen

Das folgende Beispiel zeigt, wie die Zeichenfolgenlänge einer Spalte abgerufen wird:

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

Bei breiten Zeichen lautet die Syntax wie folgt:

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

Siehe auch

- ULResultSet.GetString-Methode [UltraLite C++] auf Seite 184

GetStringLength(ul_column_num)-Methode

Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Die Anzahl von Byte oder Zeichen (ohne Nullabschlusszeichen), die erforderlich sind, um die von einer der GetString-Methoden zurückgegebene Zeichenfolge zu speichern.

Bemerkungen

Das folgende Beispiel zeigt, wie die Zeichenfolgenlänge einer Spalte abgerufen wird:

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

Bei breiten Zeichen lautet die Syntax wie folgt:

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

Siehe auch

- ULResultSet.GetString-Methode [UltraLite C++] auf Seite 184

IsNull-Methode

Überprüft, ob eine Spalte NULL ist.

Überladungsliste

Name	Beschreibung
IsNull(const char *)-Methode	Überprüft, ob eine Spalte NULL ist.
IsNull(ul_column_num)-Methode	Überprüft, ob eine Spalte NULL ist.

IsNull(const char *)-Methode

Überprüft, ob eine Spalte NULL ist.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

TRUE, wenn der Wert der Spalte NULL ist.

IsNull(ul_column_num)-Methode

Überprüft, ob eine Spalte NULL ist.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

TRUE, wenn der Wert der Spalte NULL ist.

Last-Methode

Verschiebt den Cursor in die letzte Zeile.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Next-Methode

Verschiebt den Cursor eine Zeile vorwärts.

Syntax

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

Rückgabe

TRUE, wenn der Cursor erfolgreich vorwärts verschoben wurde. Auch wenn TRUE zurückgegeben wird, kann ein Fehler ausgegeben werden, wenn der Cursor erfolgreich in die nächste Zeile bewegt wird. Es könnten z.B. Konvertierungsfehler auftreten, wenn die SELECT-Ausdrücke verarbeitet werden. In diesem Fall werden auch Fehler zurückgegeben, wenn die Spaltenwerte abgerufen werden. FALSE wird zurückgegeben, wenn der Cursor nicht vorwärts bewegt werden kann. Dies kann der Fall sein, wenn keine nächste Zeile vorhanden ist. In diesem Fall wird die resultierende Cursorposition hinter der letzten Zeile gesetzt.

Previous-Methode

Verschiebt den Cursor eine Zeile zurück.

Syntax

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

Rückgabe

TRUE, wenn sich der Cursor erfolgreich um eine Zeile zurückbewegt hat. FALSE, wenn der Cursor nicht rückwärts bewegt werden kann. Die resultierende Cursorposition wird vor der ersten Zeile gesetzt.

Relative-Methode

Verschiebt den Cursor von der aktuellen Cursorposition ausgehend um die von Offset angegebene Anzahl von Zeilen.

Syntax

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

Parameter

- **offset** Die Anzahl Zeilen, um die die Position verschoben wird.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetBinary-Methode

Legt eine Spalte mit einem ul_binary-Wert fest.

Überladungsliste

Name	Beschreibung
SetBinary(const char *, p_ul_binary)-Methode	Legt eine Spalte mit einem ul_binary-Wert fest.
SetBinary(ul_column_num, p_ul_binary)-Methode	Legt eine Spalte mit einem ul_binary-Wert fest.

SetBinary(const char *, p_ul_binary)-Methode

Legt eine Spalte mit einem ul_binary-Wert fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der ul_binary-Wert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetBinary(ul_column_num, p_ul_binary)-Methode

Legt eine Spalte mit einem ul_binary-Wert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der ul_binary-Wert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetDateTime-Methode

Legt eine Spalte mit einem DECL_DATETIME-Wert fest.

Überladungsliste

Name	Beschreibung
SetDateTime(const char *, DECL_DATETIME *)-Methode	Legt eine Spalte mit einem DECL_DATETIME-Wert fest.
SetDateTime(ul_column_num, DECL_DATETIME *)-Methode	Legt eine Spalte mit einem DECL_DATETIME-Wert fest.

SetDateTime(const char *, DECL_DATETIME *)-Methode

Legt eine Spalte mit einem DECL_DATETIME-Wert fest.

Syntax

```
public virtual bool SetDateTime(
    const char * cname,
    DECL_DATETIME * value
)
```

Parameter

- **cname** Der Name der Spalte.
- **value** Der DECL_DATETIME-Wert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetDateTime(ul_column_num, DECL_DATETIME *)-Methode

Legt eine Spalte mit einem DECL_DATETIME-Wert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der DECL_DATETIME-Wert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetDefault-Methode

Setzt eine Spalte auf ihren Standardwert.

Überladungsliste

Name	Beschreibung
SetDefault(const char *)-Methode	Setzt eine Spalte auf ihren Standardwert.
SetDefault(ul_column_num)-Methode	Setzt eine Spalte auf ihren Standardwert.

SetDefault(const char *)-Methode

Setzt eine Spalte auf ihren Standardwert.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetDefault(ul_column_num)-Methode

Setzt eine Spalte auf ihren Standardwert.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetDouble-Methode

Legt eine Spalte mit einem double-Wert fest.

Überladungsliste

Name	Beschreibung
SetDouble(const char *, ul_double)-Methode	Legt eine Spalte mit einem double-Wert fest.
SetDouble(ul_column_num, ul_double)-Methode	Legt eine Spalte mit einem double-Wert fest.

SetDouble(const char *, ul_double)-Methode

Legt eine Spalte mit einem double-Wert fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der double-Wert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetDouble(ul_column_num, ul_double)-Methode

Legt eine Spalte mit einem double-Wert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der double-Wert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetFloat-Methode

Legt eine Spalte mit einem float-Wert fest.

Überladungsliste

Name	Beschreibung
SetFloat(const char *, ul_real)-Methode	Legt eine Spalte mit einem float-Wert fest.
SetFloat(ul_column_num, ul_real)-Methode	Legt eine Spalte mit einem float-Wert fest.

SetFloat(const char *, ul_real)-Methode

Legt eine Spalte mit einem float-Wert fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der float-Wert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetFloat(ul_column_num, ul_real)-Methode

Legt eine Spalte mit einem float-Wert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der float-Wert

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetGuid-Methode

Legt eine Spalte mit einem GUID-Wert fest.

Überladungsliste

Name	Beschreibung
SetGuid(const char *, GUID *)-Methode	Legt eine Spalte mit einem GUID-Wert fest.
SetGuid(ul_column_num, GUID *)-Methode	Legt eine Spalte mit einem GUID-Wert fest.

SetGuid(const char *, GUID *)-Methode

Legt eine Spalte mit einem GUID-Wert fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der GUID-Wert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetGuid(ul_column_num, GUID *)-Methode

Legt eine Spalte mit einem GUID-Wert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der GUID-Wert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetInt-Methode

Legt eine Spalte mit einem Ganzzahlwert fest.

Überladungsliste

Name	Beschreibung
SetInt(const char *, ul_s_long)-Methode	Legt eine Spalte mit einem Ganzzahlwert fest.
SetInt(ul_column_num, ul_s_long)-Methode	Legt eine Spalte mit einem Ganzzahlwert fest.

SetInt(const char *, ul_s_long)-Methode

Legt eine Spalte mit einem Ganzzahlwert fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der Wert der Ganzzahl mit Vorzeichen (signed integer)

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetInt(ul_column_num, ul_s_long)-Methode

Legt eine Spalte mit einem Ganzzahlwert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der Wert der Ganzzahl mit Vorzeichen (signed integer)

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetIntWithType-Methode

Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.

Überladungsliste

Name	Beschreibung
SetIntWithType(const char *, ul_s_big, ul_column_storage_type)-Methode	Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.
SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type)-Methode	Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.

SetIntWithType(const char *, ul_s_big, ul_column_storage_type)-Methode

Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der Ganzzahlwert
- **type** Der integer-Datentyp, mit dem der Wert zu behandeln ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Nachstehend finden Sie eine Liste der Ganzzahlwerte, die für den value-Parameter verwendet werden können:

- 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

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

SetIntWithType(ul_column_num, ul_s_big, ul_column_storage_type)-Methode

Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der Ganzzahlwert
- **type** Der integer-Datentyp, mit dem der Wert zu behandeln ist.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Nachstehend finden Sie eine Liste der Ganzzahlwerte, die für den value-Parameter verwendet werden können:

- 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

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

SetNull-Methode

Setzt eine Spalte auf NULL.

Überladungsliste

Name	Beschreibung
SetNull(const char *)-Methode	Setzt eine Spalte auf NULL.
SetNull(ul_column_num)-Methode	Setzt eine Spalte auf NULL.

SetNull(const char *)-Methode

Setzt eine Spalte auf NULL.

Syntax

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

Parameter

- **cname** Der Name der Spalte.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetNull(ul_column_num)-Methode

Setzt eine Spalte auf NULL.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

SetString-Methode

Legt eine Spalte mit einem Zeichenfolgenwert fest.

Überladungsliste

Name	Beschreibung
SetString(const char *, const char *, size_t)-Methode	Legt eine Spalte mit einem Zeichenfolgenwert fest.
SetString(ul_column_num, const char *, size_t)-Method	Legt eine Spalte mit einem Zeichenfolgenwert fest.

SetString(const char *, const char *, size_t)-Methode

Legt eine Spalte mit einem Zeichenfolgenwert fest.

Syntax

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

Parameter

- **cname** Der Name der Spalte.
- **value** Der Zeichenfolgenwert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.
- **len** Optional. Die Länge der Zeichenfolge in Byte oder die UL_NULL_TERMINATED_STRING-Konstante, wenn die Zeichenfolge mit Null abgeschlossen ist. Die SQLE_INVALID_PARAMETER-Konstante wird gesetzt, wenn der len-Wert größer eingestellt ist als 32 kB. Rufen Sie bei großen Zeichenfolgen stattdessen die AppendStringChunk-Methode auf.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Siehe auch

- ULResultSet.AppendStringChunk-Methode [UltraLite C++] auf Seite 169

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

Legt eine Spalte mit einem Zeichenfolgenwert fest.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **value** Der Zeichenfolgenwert. Das Übergeben von NULL ist gleichwertig mit dem Aufrufen der SetNull-Methode.
- **len** Optional. Die Länge der Zeichenfolge in Byte oder die UL_NULL_TERMINATED_STRING-Konstante, wenn die Zeichenfolge mit Null abgeschlossen ist. Die SQLE_INVALID_PARAMETER-Konstante wird gesetzt, wenn der len-Wert größer eingestellt ist als 32 kB. Rufen Sie bei großen Zeichenfolgen stattdessen die AppendStringChunk-Methode auf.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Siehe auch

- ULResultSet.AppendStringChunk-Methode [UltraLite C++] auf Seite 169

Update-Methode

Aktualisiert die aktuelle Zeile.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

UpdateBegin-Methode

Wählt den Aktualisierungsmodus zum Festlegen von Spalten aus.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Spalten im Primärschlüssel können nicht geändert werden, wenn sie im Update-Modus sind.

ULResultSetSchema-Klasse

Stellt das Schema einer UltraLite-Ergebnismenge dar.

Syntax

```
public class ULResultSetSchema
```

Abgeleitete Klassen

- ULTableSchema-Klasse [UltraLite C++] auf Seite 216

Mitglieder

Alle Mitglieder der ULResultSetSchema-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
GetColumnCount-Methode	Ruft die Anzahl der Spalten in der Ergebnismenge oder Tabelle ab.
GetColumnID-Methode	Ruft die 1-basierte Spalten-ID über den Namen ab.
GetColumnName-Methode	Ruft den Namen einer Spalte anhand ihrer 1-basierten ID ab.
GetColumnPrecision-Methode	Ruft die Gesamtstellenzahl einer nummerischen Spalte ab.
GetColumnScale-Methode	Ruft die Dezimalstellen einer nummerischen Spalte ab.
GetColumnSize-Methode	Ruft die Größe der Spalte ab.
GetColumnSQLType-Methode	Ruft den SQL-Typ einer Spalte ab.
GetColumnType-Methode	Ruft den Speicher-/Hostvariablentyp einer Spalte ab.
getConnection-Methode	Ruft das ULConnection-Objekt ab.
IsAliased-Methode	Zeigt an, ob die Spalte in einer Ergebnismenge mit einem Alias versehen wurde.

GetColumnCount-Methode

Ruft die Anzahl der Spalten in der Ergebnismenge oder Tabelle ab.

Syntax

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

Rückgabe

Die Anzahl der Spalten in der Ergebnismenge oder Tabelle.

GetColumnID-Methode

Ruft die 1-basierte Spalten-ID über den Namen ab.

Syntax

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

Parameter

- **columnName** Der Spaltenname.

Rückgabe

Gibt 0 zurück, wenn die Spalte nicht vorhanden ist, ansonsten SQLE_COLUMN_NOT_FOUND, wenn der Spaltenname nicht vorhanden ist.

GetColumnName-Methode

Ruft den Namen einer Spalte anhand ihrer 1-basierten ID ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **type** Der gewünschte Spaltennamentyp.

Rückgabe

Ein Zeiger auf einen Zeichenfolgenpuffer, der den Spaltennamen enthält, sofern gefunden. Der Zeiger zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten. Wenn die Spalte nicht vorhanden ist, wird NULL zurückgegeben und SQLE_COLUMN_NOT_FOUND gesetzt.

Bemerkungen

Je nach dem ausgewählten Typ und der Art, wie die Spalte in der SELECT-Anwendung deklariert wurde, erfolgt die Rückgabe des Spaltennamens in der Form [Tabellenname].[Spaltenname].

Der type-Parameter wird verwendet, um anzugeben, welcher Spaltennamentyp zurückgegeben werden soll.

Siehe auch

- ul_column_name_type-Enumeration [UltraLite C++] auf Seite 223

GetColumnPrecision-Methode

Ruft die Gesamtstellenzahl einer nummerischen Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

0, wenn die Spalte kein nummerischer Typ ist oder nicht vorhanden ist.
SQLE_COLUMN_NOT_FOUND wird festgelegt, wenn der Spaltenname nicht vorhanden ist.
SQLE_DATATYPE_NOT_ALLOWED wird festgelegt, wenn der Spaltentyp nicht nummerisch ist.

GetColumnScale-Methode

Ruft die Dezimalstellen einer nummerischen Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

0, wenn die Spalte kein nummerischer Typ ist oder nicht vorhanden ist.
SQLE_COLUMN_NOT_FOUND wird festgelegt, wenn der Spaltenname nicht vorhanden ist.
SQLE_DATATYPE_NOT_ALLOWED wird festgelegt, wenn der Spaltentyp nicht nummerisch ist.

GetColumnSize-Methode

Ruft die Größe der Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

0, wenn die Spalte nicht vorhanden ist oder wenn der Spaltentyp keine variable Länge hat.
SQLE_COLUMN_NOT_FOUND wird festgelegt, wenn der Spaltenname nicht vorhanden ist.
SQLE_DATATYPE_NOT_ALLOWED wird festgelegt, wenn der Spaltentyp nicht
UL_SQLTYPE_CHAR oder UL_SQLTYPE_BINARY ist.

GetColumnSQLType-Methode

Ruft den SQL-Typ einer Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

UL_SQLTYPE_BAD_INDEX, wenn die Spalte nicht vorhanden ist.

Siehe auch

- ul_column_sql_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 88

GetColumnType-Methode

Ruft den Speicher-/Hostvariablentyp einer Spalte ab.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

UL_TYPE_BAD_INDEX, wenn die Spalte nicht vorhanden ist.

Siehe auch

- ul_column_storage_type-Enumeration [UltraLite C- und Embedded SQL-Datentypen] auf Seite 90

getConnection-Methode

Ruft das ULConnection-Objekt ab.

Syntax

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

Rückgabe

Das diesem Ergebnismengenschema zugeordnete ULConnection-Objekt.

IsAliased-Methode

Zeigt an, ob die Spalte in einer Ergebnismenge mit einem Alias versehen wurde.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

TRUE, wenn die Spalte mit einem Alias versehen ist, ansonsten FALSE.

ULTable-Klasse

Stellt eine Tabelle in einer UltraLite-Datenbank dar.

Syntax

```
public class ULTable : ULResultSet
```

Basisklassen

- ULResultSet-Klasse [UltraLite C++] auf Seite 166

Mitglieder

Alle Mitglieder der ULTable-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
AfterLast-Methode	Verschiebt den Cursor hinter die letzte Zeile.
AppendByteChunk-Methode	Hängt Byte an eine Spalte an.
AppendStringChunk-Methode	Hängt einen Zeichenfolgenabschnitt an eine Spalte an.
BeforeFirst-Methode	Verschiebt den Cursor vor die erste Zeile.
Close-Methode	Zerstört dieses Objekt.
Delete-Methode	Löscht die aktuelle Zeile und verschiebt den Cursor auf die nächste gültige Zeile.
DeleteAllRows-Methode	Löscht alle Zeilen aus einer Tabelle.

Name	Beschreibung
DeleteNamed-Methode	Löscht die aktuelle Zeile und verschiebt den Cursor auf die nächste gültige Zeile.
Find-Methode	Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts nach genauen Übereinstimmungen in der Tabelle gesucht wird.
FindBegin-Methode	Bereitet einen neuen Find-Aufruf in einer Tabelle vor, indem der Suchmodus gestartet wird.
FindFirst-Methode	Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts nach genauen Übereinstimmungen in der Tabelle gesucht wird.
FindLast-Methode	Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei rückwärts nach genauen Übereinstimmungen in der Tabelle gesucht wird.
FindNext-Methode	Ruft die nächste Zeile ab, die genau mit dem Index übereinstimmt.
FindPrevious-Methode	Ruft die vorherige Zeile ab, die genau mit dem Index übereinstimmt.
First-Methode	Verschiebt den Cursor in die erste Zeile.
GetBinary-Methode	Ruft einen Wert aus einer Spalte als ul_binary-Wert ab.
GetBinaryLength-Methode	Ruft die binäre Länge des Werts einer Spalte ab.
GetByteChunk-Methode	Ruft den binären Abschnitt aus der Spalte ab.
getConnection-Methode	Ruft das Verbindungsobjekt ab.
GetDateTime-Methode	Ruft einen Wert aus einer Spalte als DECL_DATETIME ab.
GetDouble-Methode	Ruft einen Wert aus einer Spalte als double-Typ ab.
GetFloat-Methode	Ruft einen Wert aus einer Spalte als float-Typ ab.
GetGuid-Methode	Ruft einen Wert aus einer Spalte als GUID ab.
GetInt-Methode	Ruft einen Wert aus einer Spalte als integer-Typ ab.
GetIntWithType-Methode	Ruft einen Wert aus einer Spalte als den angegebenen integer-Typ ab.
GetResultSetSchema-Methode	Gibt ein Objekt zurück, das verwendet werden kann, um Informationen über die Ergebnismenge zu beziehen.

Name	Beschreibung
GetRowCount-Methode	Ruft die Anzahl der Zeilen in der Tabelle ab.
GetState-Methode	Ruft den internen Zustand des Cursors ab.
GetString-Methode	Ruft einen Wert aus einer Spalte als Zeichenfolge mit einer abschließenden Null ab.
GetStringChunk-Methode	Ruft einen Zeichenfolgenabschnitt aus der Spalte ab.
GetStringLength-Methode	Ruft die Zeichenfolgenlänge des Werts einer Spalte ab.
GetTableSchema-Methode	Gibt ein ULTableSchema-Obekt zurück, das verwendet werden kann, um Schemainformationen über die Tabelle abzurufen.
Insert-Methode	Fügt eine neue Zeile in die Tabelle ein.
InsertBegin-Methode	Wählt den Einfügemodus zum Festlegen von Spalten aus.
IsNull-Methode	Überprüft, ob eine Spalte NULL ist.
Last-Methode	Verschiebt den Cursor in die letzte Zeile.
Lookup-Methode	Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts durch die Tabelle gesucht wird.
LookupBackward-Methode	Führt einen auf dem aktuellen Index basierenden Suchvogang durch, wobei rückwärts durch die Tabelle gesucht wird.
LookupBegin-Methode	Bereitet eine neue Suche in einer Tabelle vor.
LookupForward-Methode	Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts durch die Tabelle gesucht wird.
Next-Methode	Verschiebt den Cursor eine Zeile vorwärts.
Previous-Methode	Verschiebt den Cursor eine Zeile zurück.
Relative-Methode	Verschiebt den Cursor von der aktuellen Cursorposition ausgehend um die von Offset angegebene Anzahl von Zeilen.
SetBinary-Methode	Legt eine Spalte mit einem ul_binary-Wert fest.
SetDateTime-Methode	Legt eine Spalte mit einem DECL_DATETIME-Wert fest.
SetDefault-Methode	Setzt eine Spalte auf ihren Standardwert.
SetDouble-Methode	Legt eine Spalte mit einem double-Wert fest.

Name	Beschreibung
SetFloat-Methode	Legt eine Spalte mit einem float-Wert fest.
SetGuid-Methode	Legt eine Spalte mit einem GUID-Wert fest.
SetInt-Methode	Legt eine Spalte mit einem Ganzzahlwert fest.
SetIntWithType-Methode	Legt eine Spalte mit einem Ganzzahlwert des angegebenen integer-Typs fest.
SetNull-Methode	Setzt eine Spalte auf NULL.
SetString-Methode	Legt eine Spalte mit einem Zeichenfolgenwert fest.
TruncateTable-Methode	Kürzt die Tabelle und aktiviert STOP SYNCHRONIZATION DELETE vorübergehend.
Update-Methode	Aktualisiert die aktuelle Zeile.
UpdateBegin-Methode	Wählt den Aktualisierungsmodus zum Festlegen von Spalten aus.

DeleteAllRows-Methode

Löscht alle Zeilen aus einer Tabelle.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE. Beispiel: FALSE wird zurückgegeben, wenn die Tabelle nicht geöffnet ist oder ein SQL-Fehler aufgetreten ist.

Bemerkungen

In einigen Anwendungen kann es nützlich sein, alle Zeilen aus einer Tabelle zu löschen, bevor eine neue Datenmenge in die Tabelle geladen wird. Wenn die StopSync-Eigenschaft für diese Verbindung eingestellt ist, werden die gelöschten Zeilen nicht synchronisiert.

Hinweis
Nicht festgeschriebene Einfügungen von anderen Verbindungen werden nicht gelöscht. Sie werden ebenfalls nicht gelöscht, wenn die andere Verbindung nach dem Aufrufen der DeleteAllRows-Methode ein Rollback durchführt.

Wenn diese Tabelle ohne Index geöffnet wurde, wird sie als schreibgeschützt angesehen und es können keine Daten gelöscht werden.

Find-Methode

Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts nach genauen Übereinstimmungen in der Tabelle gesucht wird.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes die Anzahl an Spalten, die bei der Suche verwendet werden soll.

Rückgabe

Wenn keine Zeile dem Indexwert entspricht, wird die Cursorposition hinter der letzten Zeile gesetzt und die Methode gibt FALSE zurück.

Bemerkungen

Um den zu suchenden Wert anzugeben, legen Sie den Spaltenwert für die einzelnen Spalten im Index fest. Der Cursor wird in die erste Zeile positioniert, die genau mit dem Indexwert übereinstimmt.

FindBegin-Methode

Bereitet einen neuen Find-Aufruf in einer Tabelle vor, indem der Suchmodus gestartet wird.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Es können nur Spalten in dem Index eingestellt werden, mit dem die Tabelle geöffnet wurde. Diese Methode kann nicht aufgerufen werden, wenn die Tabelle ohne Index geöffnet wurde.

FindFirst-Methode

Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts nach genauen Übereinstimmungen in der Tabelle gesucht wird.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes die Anzahl an Spalten, die bei der Suche verwendet werden soll.

Rückgabe

Wenn keine Zeile dem Indexwert entspricht, wird die Cursorposition hinter der letzten Zeile gesetzt und die Methode gibt FALSE zurück.

Bemerkungen

Um den zu suchenden Wert anzugeben, legen Sie den Spaltenwert für die einzelnen Spalten im Index fest. Der Cursor wird in die erste Zeile positioniert, die genau mit dem Indexwert übereinstimmt.

FindLast-Methode

Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei rückwärts nach genauen Übereinstimmungen in der Tabelle gesucht wird.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes die Anzahl an Spalten, die bei der Suche verwendet werden soll.

Rückgabe

Wenn keine Zeile dem Indexwert entspricht, wird die Cursorposition vor der ersten Zeile gesetzt und die Methode gibt FALSE zurück.

Bemerkungen

Um den zu suchenden Wert anzugeben, legen Sie den Spaltenwert für die einzelnen Spalten im Index fest. Der Cursor wird in die erste Zeile positioniert, die genau mit dem Indexwert übereinstimmt.

FindNext-Methode

Ruft die nächste Zeile ab, die genau mit dem Index übereinstimmt.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes die Anzahl an Spalten, die bei der Suche verwendet werden soll.

Rückgabe

FALSE, wenn keine weiteren Zeilen mit dem Index übereinstimmen. In diesem Fall wird der Cursor hinter der letzten Zeile positioniert.

FindPrevious-Methode

Ruft die vorherige Zeile ab, die genau mit dem Index übereinstimmt.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes die Anzahl an Spalten, die bei der Suche verwendet werden soll.

Rückgabe

FALSE, wenn keine weiteren Zeilen mit dem Index übereinstimmen. In diesem Fall wird der Cursor vor der ersten Zeile positioniert.

GetTableSchema-Methode

Gibt ein ULTableSchema-Obekt zurück, das verwendet werden kann, um Schemainformationen über die Tabelle abzurufen.

Syntax

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

Rückgabe

Ein ULTableSchema-Objekt, das verwendet werden kann, um Schemainformationen über die Ergebnismenge zu beziehen.

Insert-Methode

Fügt eine neue Zeile in die Tabelle ein.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

InsertBegin-Methode

Wählt den Einfügemodus zum Festlegen von Spalten aus.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Während eines Einfügevorgangs werden alle Spalten auf ihren Standardwert gesetzt, sofern nicht über Set-Methodenaufrufe ein anderer Wert angegeben wird.

Lookup-Methode

Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts durch die Tabelle gesucht wird.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes ist dies die Anzahl der Spalten, die bei der Suche verwendet werden soll.

Rückgabe

FALSE, wenn die resultierende Cursorposition hinter der letzten Zeile gesetzt wird.

Bemerkungen

Um den zu suchenden Wert anzugeben, legen Sie den Spaltenwert für die einzelnen Spalten im Index fest. Der Cursor wird in die letzten Zeile positioniert, die mit dem Indexwert übereinstimmt oder kleiner als dieser Wert ist. Bei zusammengesetzten Indizes gibt der ncols-Parameter die Anzahl an Spalten an, die bei der Suche verwendet werden soll.

LookupBackward-Methode

Führt einen auf dem aktuellen Index basierenden Suchvogang durch, wobei rückwärts durch die Tabelle gesucht wird.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes ist dies die Anzahl der Spalten, die bei der Suche verwendet werden soll.

Rückgabe

FALSE, wenn die resultierende Cursorposition vor der ersten Zeile gesetzt wird.

Bemerkungen

Um den zu suchenden Wert anzugeben, legen Sie den Spaltenwert für die einzelnen Spalten im Index fest. Der Cursor wird in die letzten Zeile positioniert, die mit dem Indexwert übereinstimmt oder kleiner als dieser Wert ist. Bei zusammengesetzten Indizes gibt der ncols-Parameter die Anzahl an Spalten an, die bei der Suche verwendet werden soll.

LookupBegin-Methode

Bereitet eine neue Suche in einer Tabelle vor.

Syntax

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

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Es können nur Spalten in dem Index eingestellt werden, mit dem die Tabelle geöffnet wurde. Wenn die Tabelle ohne Index geöffnet wurde, kann diese Methode nicht aufgerufen werden.

LookupForward-Methode

Führt einen auf dem aktuellen Index basierenden Suchvorgang durch, wobei vorwärts durch die Tabelle gesucht wird.

Syntax

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

Parameter

- **ncols** Bei zusammengesetzten Indizes ist dies die Anzahl der Spalten, die bei der Suche verwendet werden soll.

Rückgabe

FALSE, wenn die resultierende Cursorposition hinter der letzten Zeile gesetzt wird.

Bemerkungen

Um den zu suchenden Wert anzugeben, legen Sie den Spaltenwert für die einzelnen Spalten im Index fest. Der Cursor wird in die letzten Zeile positioniert, die mit dem Indexwert übereinstimmt oder kleiner als dieser Wert ist. Bei zusammengesetzten Indizes gibt der ncols-Parameter die Anzahl an Spalten an, die bei der Suche verwendet werden soll.

TruncateTable-Methode

Kürzt die Tabelle und aktiviert STOP SYNCHRONIZATION DELETE vorübergehend.

Syntax

```
public virtual bool TruncateTable()
```

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

ULTableSchema-Klasse

Stellt das Schema einer UltraLite-Tabelle dar.

Syntax

```
public class ULTableSchema : ULResultSetSchema
```

Basisklassen

- ULResultSetSchema-Klasse [UltraLite C++] auf Seite 202

Mitglieder

Alle Mitglieder der ULTableSchema-Klasse, einschließlich aller geerbten Mitglieder.

Name	Beschreibung
Close-Methode	Zerstört dieses Objekt.
GetColumnCount-Methode	Ruft die Anzahl der Spalten in der Ergebnismenge oder Tabelle ab.
GetColumnDefault-Methode	Ruft den Standardwert für die Spalte ab, falls vorhanden.
GetColumnDefaultType-Methode	Ruft den Typ des Spaltenstandardwerts ab.
GetColumnID-Methode	Ruft die 1-basierte Spalten-ID über den Namen ab.
GetColumnName-Methode	Ruft den Namen einer Spalte anhand ihrer 1-basierten ID ab.
GetColumnPrecision-Methode	Ruft die Gesamtstellenzahl einer nummerischen Spalte ab.
GetColumnScale-Methode	Ruft die Dezimalstellen einer nummerischen Spalte ab.
GetColumnSize-Methode	Ruft die Größe der Spalte ab.
GetColumnSQLType-Methode	Ruft den SQL-Typ einer Spalte ab.
GetColumnType-Methode	Ruft den Speicher-/Hostvariablentyp einer Spalte ab.
getConnection-Methode	Ruft das ULConnection-Objekt ab.

Name	Beschreibung
GetGlobalAutoincPartitionSize-Methode	Ruft die Partitionsgröße ab.
GetIndexCount-Methode	Ruft die Anzahl der Indizes in der Tabelle ab.
GetIndexSchema-Methode	Ruft das Schema für einen Index anhand des Namens ab.
GetName-Methode	Ruft den Namen der Tabelle ab.
GetNextIndex-Methode	Ruft den nächsten Index (Schema) in der Tabelle ab.
GetOptimalIndex-Methode	Ermittelt den besten Index für die Suche nach einem Spaltenwert.
GetPrimaryKey-Methode	Ruft den Primärschlüssel für die Tabelle ab.
GetPublicationPredicate-Methode	Ruft das Publikationsprädikat als Zeichenfolge ab.
GetTableSyncType-Methode	Ruft den Tabellensynchronisationstyp ab.
InPublication-Methode	Prüft, ob die Tabelle in der genannten Publikation enthalten ist.
IsAliased-Methode	Zeigt an, ob die Spalte in einer Ergebnismenge mit einem Alias versehen wurde.
IsColumnInIndex-Methode	Überprüft, ob die Spalte im benannten Index enthalten ist.
IsColumnNullable-Methode	Prüft, ob die angegebene Spalte nullwertfähig ist.

Close-Methode

Zerstört dieses Objekt.

Syntax

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

GetColumnDefault-Methode

Ruft den Standardwert für die Spalte ab, falls vorhanden.

Syntax

```
public virtual const char * GetColumnDefault(ul_column_num cid)
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Der Standardwert. Eine leere Zeichenfolge wird zurückgegeben, wenn die Spalte keinen Standardwert hat. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

GetColumnDefaultType-Methode

Ruft den Typ des Spaltenstandardwerts ab.

Syntax

```
public virtual ul_column_default_type GetColumnDefaultType(
    ul_column_num cid
)
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Der Typ des Spaltenstandardwerts.

Siehe auch

- ul_column_default_type-Enumeration [UltraLite C++] auf Seite 222

GetGlobalAutoincPartitionSize-Methode

Ruft die Partitionsgröße ab.

Syntax

```
public virtual bool GetGlobalAutoincPartitionSize(
    ul_column_num cid,
    ul_u_big * size
)
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **size** Ein Ausgabeparameter. Die Partitionsgröße für die Spalte Alle global autoincrement-Spalten in einer gegebenen Tabelle verwenden dieselbe global autoincrement-Partition.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

GetIndexCount-Methode

Ruft die Anzahl der Indizes in der Tabelle ab.

Syntax

```
public virtual ul_index_num GetIndexCount()
```

Rückgabe

Die Anzahl der Indizes in der Tabelle

Bemerkungen

Index-IDs und Indexanzahl können sich während eines Schema-Upgrades ändern. Um einen Index richtig zu identifizieren, rufen Sie ihn anhand des Namens auf oder aktualisieren nach jedem Schema-Upgrade alle IDs und Zähler im Cache.

GetIndexSchema-Methode

Ruft das Schema für einen Index anhand des Namens ab.

Syntax

```
public virtual ULIndexSchema * GetIndexSchema(const char * indexName)
```

Parameter

- **indexName** Der Name des Indexes.

Rückgabe

Ein ULIndexSchema-Objekt für den angegebenen Index oder NULL, wenn ein solches nicht vorhanden ist.

GetName-Methode

Ruft den Namen der Tabelle ab.

Syntax

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

Rückgabe

Der Name der Tabelle. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

GetNextIndex-Methode

Ruft den nächsten Index (Schema) in der Tabelle ab.

Syntax

```
public virtual ULIndexSchema * GetNextIndex(ul_index_iter * iter)
```

Parameter

- **iter** Ein Zeiger auf die Wiederholervariable.

Rückgabe

Ein ULIndexSchema-Objekt oder NULL, wenn die Wiederholung abgeschlossen ist.

Bemerkungen

Initialisieren Sie den iter-Wert vor dem ersten Aufruf auf die ul_index_iter_start-Konstante.

Siehe auch

- ul_index_iter_start-Variable [UltraLite C++] auf Seite 226

GetOptimalIndex-Methode

Ermittelt den besten Index für die Suche nach einem Spaltenwert.

Syntax

```
public virtual const char * GetOptimalIndex(ul_column_num cid)
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

Der Name des Indexes oder NULL, wenn die Spalte nicht indexiert ist. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

GetPrimaryKey-Methode

Ruft den Primärschlüssel für die Tabelle ab.

Syntax

```
public virtual ULIndexSchema * GetPrimaryKey()
```

Rückgabe

Ein ULIndexSchema-Objekt für den Primärschlüssel der Tabelle.

GetPublicationPredicate-Methode

Ruft das Publikationsprädikat als Zeichenfolge ab.

Syntax

```
public virtual const char * GetPublicationPredicate(
    const char * pubName
)
```

Parameter

- **pubName** Der Name der Publikation

Rückgabe

Die Publikationsprädikat-Zeichenfolge für die angegebene Publikation. Dieser Wert zeigt auf einen statischen Puffer, dessen Inhalt durch jeden nachfolgenden UltraLite-Aufruf geändert werden kann. Daher müssen Sie eine Kopie des Werts erstellen, wenn Sie ihn aufbewahren möchten.

GetTableSyncType-Methode

Ruft den Tabellensynchronisationstyp ab.

Syntax

```
public virtual ul_table_sync_type GetTableSyncType()
```

Rückgabe

Der Tabellensynchronisationstyp.

Bemerkungen

Diese Methode zeigt an, wie die Tabelle an der Synchronisation teilnimmt, und wird festgelegt, wenn die Tabelle mit der SYNCHRONIZE-Integritätsregelklausel der CREATE TABLE-Anweisung erstellt wird.

Siehe auch

- ul_table_sync_type-Enumeration [UltraLite C++] auf Seite 225

InPublication-Methode

Prüft, ob die Tabelle in der genannten Publikation enthalten ist.

Syntax

```
public virtual bool InPublication(const char * pubName)
```

Parameter

- **pubName** Der Name der Publikation

Rückgabe

TRUE, wenn die Tabelle in der Publikation enthalten ist, ansonsten FALSE.

IsColumnInIndex-Methode

Überprüft, ob die Spalte im benannten Index enthalten ist.

Syntax

```
public virtual bool IsColumnInIndex(
    ul_column_num cid,
    const char * indexName
)
```

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.
- **indexName** Der Name des Indexes.

Rückgabe

TRUE, wenn die Spalte im Index enthalten ist, ansonsten FALSE.

IsColumnNullable-Methode

Prüft, ob die angegebene Spalte nullwertfähig ist.

Syntax

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

Parameter

- **cid** Die 1-basierte Ordinalzahl der Spaltennummer.

Rückgabe

TRUE, wenn die Spalte nullwertfähig ist, ansonsten FALSE.

ul_column_default_type-Enumeration

Kennzeichnet einem Spaltenstandardtyp.

Syntax

```
public enum ul_column_default_type
```

Mitglieder

Mitgliedsname	Beschreibung
ul_column_default_none	Die Spalte hat keinen Standardwert.
ul_column_default_autoincrement	Der Spaltenstandardwert ist AUTOINCREMENT.

Mitgliedsname	Beschreibung
ul_column_default_global_autoincrement	Der Spaltenstandardwert ist GLOBAL AUTOINCREMENT.
ul_column_default_current_timestamp	Der Spaltenstandardwert ist CURRENT TIMESTAMP.
ul_column_default_current_utc_timestamp	Der Spaltenstandardwert ist CURRENT UTC TIMESTAMP.
ul_column_default_current_time	Der Spaltenstandardwert ist CURRENT TIME.
ul_column_default_current_date	Der Spaltenstandardwert ist CURRENT DATE.
ul_column_default_newid	Der Spaltenstandardwert ist die NEWID().
ul_column_default_other	Der Spaltenstandardwert ist eine benutzerdefinierte Konstante.

Siehe auch

- ULTableSchema.GetColumnDefaultType-Methode [UltraLite C++] auf Seite 218

ul_column_name_type-Enumeration

Gibt Werte an, die steuern, wie ein Spaltenname beim Beschreiben einer Ergebnismenge abgerufen wird.

Syntax

```
public enum ul_column_name_type
```

Mitglieder

Mitgliedsname	Beschreibung
ul_name_type_sql	Gibt bei SELECT-Anweisungen Alias oder Korrelationsname zurück. Gibt bei Tabellen den Spaltennamen zurück.
ul_name_type_sql_column_only	Gibt bei SELECT-Anweisungen Alias oder Korrelationsname zurück und schließt Tabellennamen aus, die angegeben wurden. Gibt bei Tabellen den Spaltennamen zurück.

Mitgliedsname	Beschreibung
ul_name_type_base_table	Gibt den Namen der Basistabelle zurück, wenn dieser ermittelt werden kann. Wenn die Tabelle im Datenbankschema nicht vorhanden ist, wird eine leere Zeichenfolge zurückgegeben.
ul_name_type_base_column	Gibt den Namen der zugrunde liegenden Spalte zurück, wenn dieser ermittelt werden kann. Wenn die Spalte im Datenbankschema nicht vorhanden ist, wird eine leere Zeichenfolge zurückgegeben.
ul_name_type_qualified	Gibt den qualifizierten Namen der zugrunde liegenden Spalte zurück, wenn er ermittelt werden kann, bei Verwendung zusammen mit der ULResultSetSchema.GetColumnName-Methode. Der zurückgegebene Name kann einer der folgenden Werte sein und wird in dieser Reihenfolge ermittelt: 1. Die dargestellte korrelierte Tabelle 2. Der Name der dargestellten Tabellenspalte 3. Der Aliasname der Spalte 4. Eine leere Zeichenfolge
ul_name_type_base	Gibt an, dass ein mit dem Tabellennamen qualifizierter Spaltenname zurückgegeben werden muss, wenn die GetColumnName-Methode verwendet wird. Wenn der abgerufene Spaltenname in der Abfrage einer Basistabelle zugeordnet ist, wird der Name der Basistabelle als Qualifizierer der Spalte verwendet (d.h., der base_table_name.column_name-Wert wird zurückgegeben). Wenn der abgerufene Spaltenname sich auf eine Spalte in einer korrelierten Tabelle in der Abfrage bezieht, wird der Korrelationsname als Qualifizierer der Spalte verwendet (d.h., der correl_table_name.col_name-Wert wird zurückgegeben). Wenn die Spalte einen Alias hat, wird der qualifizierte Name der Spalte, die als Alias zurückgegeben wird. Der Alias ist kein Teil des qualifizierten Namens. Sonst wird eine leere Zeichenfolge zurückgegeben.

Siehe auch

- ULResultSetSchema.GetColumnName-Methode [UltraLite C++] auf Seite 204

ul_index_flag-Enumeration

Flags (Bitfelder), die Eigenschaften eines Indexes identifizieren.

Syntax

```
public enum ul_index_flag
```

Mitglieder

Mitgliedsname	Beschreibung
ul_index_flag_primary_key	Der Index ist ein Primärschlüssel.
ul_index_flag_unique_key	Der Index ist ein Primärschlüssel oder ein Index, der für eine Eindeutigkeits-Integritätsregel (NULL nicht zulässig) erstellt wurde.
ul_index_flag_unique_index	Der Index wurde mit dem UNIQUE-Parameter erstellt (oder ist ein Primärschlüssel).
ul_index_flag_foreign_key	Der Index ist ein Fremdschlüssel.
ul_index_flag_foreign_key_nullable	Der Fremdschlüssel lässt NULL zu.
ul_index_flag_foreign_key_check_on_commit	Prüfungen der referenziellen Integrität werden nach dem Festschreiben durchgeführt (und nicht bei Insert/Update).

Siehe auch

- ULIndexSchema.GetIndexFlags-Methode [UltraLite C++] auf Seite 154

ul_table_sync_type-Enumeration

Kennzeichnet einen Tabellensynchronisationstyp.

Syntax

```
public enum ul_table_sync_type
```

Mitglieder

Mitgliedsname	Beschreibung
ul_table_sync_on	Alle geänderten Zeilen werden synchronisiert. Dies ist das Standardverhalten. Dieser Initialisierer entspricht der SYNCHRONIZE ON-Klausel in einer CREATE TABLE-Anweisung.

Mitgliedsname	Beschreibung
ul_table_sync_off	Tabelle wird nie synchronisiert. Dieser Initialisierer entspricht der SYNCHRONIZE OFF-Klausel in einer CREATE TABLE-Anweisung.
ul_table_sync_upload_all_rows	Es werden immer alle Zeilen hochgeladen, auch nicht geänderte Zeilen. Dieser Initialisierer entspricht der SYNCHRONIZE ALL-Klausel in einer CREATE TABLE-Anweisung.
ul_table_sync_download_only	Änderungen werden nie hochgeladen. Dieser Initialisierer entspricht der SYNCHRONIZE DOWNLOAD-Klausel in einer CREATE TABLE-Anweisung.

Siehe auch

- ULTableSchema.GetTableSyncType-Methode [UltraLite C++] auf Seite 221

UL_BLOB_CONTINUE-Variable

Wird beim Lesen von Daten mit der Methode ULResultSet.GetStringChunk oder ULResultSet.GetByteChunk verwendet.

Syntax

```
#define UL_BLOB_CONTINUE
```

Bemerkungen

Dieser Wert zeigt an, dass der zu lesende Datenabschnitt bis zu der Stelle fortgesetzt werden soll, an der der letzte Abschnitt gelesen wurde.

Siehe auch

- ULResultSet.GetStringChunk-Methode [UltraLite C++] auf Seite 185
- ULResultSet.GetByteChunk-Methode [UltraLite C++] auf Seite 174

ul_index_iter_start-Variable

Von der GetNextIndex-Methode verwendet, um die Indexiteration in einer Tabelle zu initialisieren.

Syntax

```
#define ul_index_iter_start
```

Siehe auch

- ULTableSchema.GetNextIndex-Methode [UltraLite C++] auf Seite 219

ul_publication_iter_start-Variable

Wird von der GetNextPublication-Methode verwendet, um die Publikationsiteration in einer Datenbank zu initialisieren.

Syntax

```
#define ul_publication_iter_start
```

Siehe auch

- ULDatabaseSchema.GetNextPublication-Methode [UltraLite C++] auf Seite 146

ul_table_iter_start-Variable

Wird von der GetNextTable-Methode verwendet, um die Tabelleniteration in einer Datenbank zu initialisieren.

Syntax

```
#define ul_table_iter_start
```

Siehe auch

- ULDatabaseSchema.GetNextTable-Methode [UltraLite C++] auf Seite 146

UltraLite Embedded SQL API-Referenz

Dieser Abschnitt listet die Funktionen auf, die die UltraLite-Funktionalität in Embedded SQL unterstützen.

Allgemeine Informationen zu verfügbaren SQL-Anweisungen finden Sie unter „UltraLite C++-Anwendungsentwicklung mit Embedded SQL“ auf Seite 35.

Verwenden Sie den Befehl EXEC SQL INCLUDE SQLCA, um die Prototypen für die Funktionen in diesem Kapitel einzubinden.

Headerdateien

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

db_fini-Methode

Gibt von der UltraLite-Laufzeitbibliothek benutzte Ressourcen frei

Syntax

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

Rückgabe

- 0, wenn während der Verarbeitung ein Fehler auftritt. Der Fehlererror wird in SQLCA gesetzt.
- Nicht 0, wenn kein Fehler auftritt.

Bemerkungen

Nachdem Sie db_fini aufgerufen haben, sind keine weiteren UltraLite-Bibliotheksaufrufe und keine weiteren Embedded SQL-Befehle erlaubt.

Rufen Sie db_fini einmal für jeden benutzten SQLCA auf.

Siehe auch

- „db_init-Methode“ auf Seite 228

db_init-Methode

Initialisiert die UltraLite-Laufzeitbibliothek

Syntax

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

Rückgabe

- 0, wenn während der Verarbeitung ein Fehler auftritt (z.B. während der Initialisierung des beständigen Speichers). Der Fehlererror wird in SQLCA gesetzt.
- Nicht 0, wenn kein Fehler auftritt. Sie können mit der Verwendung der Embedded SQL-Befehle und -Funktionen beginnen.

Bemerkungen

Sie müssen diese Funktion aufrufen, bevor Sie einen weiteren UltraLite-Bibliotheksaufruf oder einen Embedded SQL-Befehl ausführen.

Normalerweise rufen Sie diese Funktion nur einmal auf und übergeben die Adresse der globalen sqlca-Variablen (wie in der Header-Datei *sqlca.h* definiert). Wenn Ihre Anwendung mehrere Ausführungspfade enthält, können Sie mehrere db_init-Aufrufe verwenden, sofern jeder Aufruf einen eigenen sqlca-Zeiger besitzt. Dieser separate SQLCA-Zeiger kann benutzerdefiniert oder auch ein globaler SQLCA-Bereich sein, der mit db_fini freigegeben wurde.

In Anwendungen mit mehreren Threads muss jeder Thread db_init aufrufen, um eigene SQLCA zu erhalten. Nachfolgende Verbindungen und Transaktionen, die diesen SQLCA-Kommunikationsbereich verwenden, müssen mit einem einzigen Thread ausgeführt werden.

Durch die Initialisierung des SQLCA-Bereichs werden auch alle Einstellungen aus zuvor aufgerufenen ULEnable-Funktionen zurückgesetzt. Wenn Sie einen SQLCA-Bereich neu initialisieren, müssen Sie alle von der Anwendung erforderten ULEnable-Funktionen ausführen.

Siehe auch

- „db_fini-Methode“ auf Seite 227

MLFTEnableRsaE2ee-Methode

Ermöglicht Ihnen das Angeben der RSA-Ende-zu-Ende-Verschlüsselungsfunktion.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

MLFTEnableRsaEncryption-Methode

Ermöglicht Ihnen das Angeben der RSA-Verschlüsselungsfunktion.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

MLFTEnableRsaFipsE2ee-Methode

Ermöglicht Ihnen das Angeben der RSAFIPS-Ende-zu-Ende-Verschlüsselungsfunktion.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

MLFTEnableRsaFipsEncryption-Methode

Ermöglicht Ihnen das Angeben der RSAFIPS-Verschlüsselungsfunktion.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

MLFTEnableZlibCompression-Methode

Ermöglicht Ihnen das Angeben der ZLIB-Komprimierungsfunktion.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

MLFileDownload-Methode

Lädt eine Datei von einem MobiLink-Server über die MobiLink-Schnittstelle herunter

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Bemerkungen

Sie müssen den Quellspeicherort der zu übertragenden Datei angeben. Dieser Speicherort muss als Verzeichnis eines MobiLink-Benutzers auf dem MobiLink-Server (oder im Standardverzeichnis auf diesem Server) festgelegt sein. Sie können auch den gewünschten Zielspeicherort und den Namen der Datei angeben.

Sie können z.B. Ihre Anwendung so programmieren, dass sie eine neue Datenbank oder eine Ersatzdatenbank vom MobiLink-Server herunterlädt. Sie können die Datei für bestimmte Benutzer anpassen, da der erste Speicherort, der durchsucht wird, das Unterverzeichnis eines bestimmtes Benutzers

ist. Sie können auch eine Standarddatei im Stammverzeichnis auf dem Server verwalten, da dieser Speicherort verwendet wird, wenn die betreffende Datei im Benutzerverzeichnis nicht gefunden wird.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

Beispiel

Das folgende Beispiel zeigt, wie Sie die MLFileDownload-Methode verwenden:

```
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 );
```

MLFileUpload-Methode

Lädt eine Datei von einem MobiLink-Server über die MobiLink-Schnittstelle hoch

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Bemerkungen

Sie müssen den Quellspeicherort der zu übertragenden Datei angeben. Dieser Speicherort muss als Verzeichnis eines MobiLink-Benutzers auf dem MobiLink-Server (oder im Standardverzeichnis auf diesem Server) festgelegt sein. Sie können auch den gewünschten Zielspeicherort und den Namen der Datei angeben.

Sie können z.B. Ihre Anwendung so programmieren, dass sie eine neue Datenbank oder eine Ersatzdatenbank vom MobiLink-Server hochlädt. Sie können die Datei für bestimmte Benutzer anpassen, da der erste Speicherort, der durchsucht wird, das Unterverzeichnis eines bestimmtes Benutzers ist. Sie können auch eine Standarddatei im Stammverzeichnis auf dem Server verwalten, da dieser Speicherort verwendet wird, wenn die betreffende Datei im Benutzerverzeichnis nicht gefunden wird.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

Beispiel

Das folgende Beispiel zeigt, wie Sie die MLFileUpload-Methode verwenden:

```
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 );
```

MLFiniFileTransferInfo-Methode

Finalisiert Ressourcen, die in der ml_file_transfer_info-Struktur zugewiesen werden, wenn diese initialisiert wird.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Bemerkungen

Diese Methode sollte aufgerufen werden, nachdem der Upload/Download der Datei abgeschlossen ist.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

MLInitFileTransferInfo-Methode

Initialisiert die ml_file_transfer_info-Struktur.

Syntax

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

Parameter

- **info** Eine Struktur mit den Informationen für die Dateiübertragung.

Bemerkungen

Diese Methode muss vor dem Starten des Uploads/Downloads aufgerufen werden.

Siehe auch

- ml_file_transfer_info-Struktur [UltraLite Embedded SQL] auf Seite 267

ULCancelGetNotification-Methode

Bricht ausstehende getNotification-Aufrufe in allen Warteschlangen ab, die mit dem angegebenen Namen übereinstimmen.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **queue_name** Der Name der Warteschlange.

Rückgabe

Die Anzahl der betroffenen Warteschlangen (nicht notwendigerweise die Anzahl der blockierten Lesevorgänge).

ULChangeEncryptionKey-Methode

Ändert die Verschlüsselung für eine UltraLite-Datenbank

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **new_key** Der neue Chiffrierschlüssel.

Bemerkungen

Anwendungen, die diese Methode aufrufen, müssen zunächst sicherstellen, dass der Benutzer entweder die Datenbank synchronisiert oder eine zuverlässige Sicherungskopie der Datenbank erstellt hat. Die zuverlässige Sicherung der Datenbank ist wichtig, da diese Methode vollständig ausgeführt werden muss. Wenn Sie den Datenbank-Chiffrierschlüssel ändern, wird jede Zeile in der Datenbank zuerst mit dem alten Schlüssel entschlüsselt und dann mit dem neueren Schlüssel verschlüsselt, bevor die Zeile neu geschrieben wird. Dieser Vorgang kann nicht rückgängig gemacht werden. Wenn der Änderungsvorgang

der Verschlüsselung nicht abgeschlossen wird, hat die Datenbank einen ungültigen Status und Sie können nicht mehr auf sie zugreifen.

ULCheckpoint-Methode

Führt einen Checkpoint-Vorgang aus, der alle noch festzuschreibenden Transaktionen in der Datenbank bereinigt.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Bemerkungen

Aktuell ausgeführte Transaktionen werden durch Aufrufen dieser Methode nicht festgeschrieben. Diese Methode wird in Verbindung mit dem Verzögern automatischer Transaktions-Checkpoints als Performance-Verbesserung eingesetzt.

Diese Methode stellt sicher, dass alle noch festzuschreibenden Transaktionen in die Datenbank geschrieben wurden.

ULCountUploadRows-Methode

Ermittelt die Anzahl der Zeilen, die für die Synchronisation übertragen werden sollen

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQL-Bereich
- **pub_list** Eine Zeichenfolge mit einer kommagetrennten Liste von Publikationen, die geprüft werden sollen. Eine leere Zeichenfolge (der UL_SYNC_ALL-Makro) bedeutet alle Tabellen, außer den als "no sync" markierten. Eine Zeichenfolge, die nur ein Sternchen enthält (der UL_SYNC_ALL_PUBS-Makro), bedeutet alle Tabellen, die in einer Publikation referenziert werden. Einige Tabellen sind in keiner Publikation referenziert und werden daher nicht einbezogen, wenn die pub-list-Zeichenfolge "*" ist.
- **threshold** Ermittelt die maximale Anzahl der zu zählenden Zeilen, wodurch die Zeitdauer des Aufrufs begrenzt wird. Der Schwellenwert 0 entspricht keiner Beschränkung (d.h., die Methode zählt alle zu synchronisierenden Zeilen) und der Schwellenwert 1 kann verwendet werden, um schnell zu ermitteln, ob Zeilen synchronisiert werden müssen.

Rückgabe

Die Anzahl der Zeilen, die synchronisiert werden müssen, entweder in einer angegebenen Gruppe von Publikationen oder in der gesamten Datenbank.

Bemerkungen

Verwenden Sie diese Methode, um Benutzer zur Synchronisation aufzufordern oder um festzulegen, wann die automatische Hintergrundsynchronisation erfolgen soll.

Der folgende Aufruf überprüft die gesamte Datenbank auf die Gesamtzahl der zu synchronisierenden Zeilen:

```
count = ULCountUploadRows( sqlca, UL_SYNC_ALL, 0 );
```

Der folgende Aufruf überprüft die Publikationen PUB1 und PUB2 auf ein Maximum von 1000 Zeilen:

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

Der folgende Aufruf prüft, ob in den Publikationen PUB1 und PUB2 Zeilen synchronisiert werden müssen:

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

ULCreateDatabase-Methode

Erstellt eine UltraLite-Datenbank

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den initialisierten SQLCA-Bereich
- **connect_parms** Eine durch Semikola getrennte Zeichenfolge von Verbindungsparametern, die in der Form von Schlüsselwort=Wert-Paaren festgelegt werden. Die Verbindungszeichenfolge muss den Namen der Datenbank enthalten. Es handelt sich um dieselben Parameter, die beim Herstellen der Verbindung mit einer Datenbank angegeben werden können.
- **create_parms** Eine durch Semikola getrennte Zeichenfolge von Erstellungsparametern, festgelegt als Schlüsselwort=Wert-Paare, z.B. "page_size=2048;obfuscate=yes".
- **reserved** Dieser Parameter ist für die zukünftige Verwendung reserviert.

Rückgabe

Gibt ul_true zurück, wenn die Datenbank erfolgreich erstellt wurde, ansonsten ul_false. Die Rückgabe von ul_false wird gewöhnlich durch einen ungültigen Dateinamen oder durch eine Verweigerung des Zugriffs verursacht.

Bemerkungen

Die Datenbank wird mit Informationen erstellt, die in zwei Parametergruppen bereitgestellt werden.

Der connect_parms-Parameter ist eine Liste von Verbindungsparametern, die immer anwendbar sind, wenn auf die Datenbank zugegriffen wird. Dazu gehören beispielsweise Dateiname, Benutzer-ID, Kennwort und optionaler Chiffrierschlüssel.

Der create_parms-Parameter ist eine Liste von Parametern, die nur zum Zeitpunkt der Datenbankerstellung relevant sind. Dazu gehören beispielsweise Verschleierung, Seitengröße sowie Uhrzeit- und Datumsformat.

Anwendungen können diese Methode nach der Initialisierung des SQLCA aufrufen.

Der folgende Code zeigt, wie Sie die ULCreateDatabase-Methode verwenden, um eine UltraLite-Datenbank mit dem Dateinamen *C:\myfile.udb* zu erstellen:

```
if( ULCreateDatabase(&sqlca
    ,UL_TEXT("DBF=C:\myfile.udb;uid=DBA;pwd=sql")
    ,ULGetCollation_1250LATIN2()
    ,UL_TEXT("obfuscate=1;page_size=8192")
    ,NULL)
{
    // success
};
```

Siehe auch

- „UltraLite-Verbindungsparameter“ [*UltraLite - Datenbankverwaltung*]
- „UltraLite-Erstellungsparameter“ [*UltraLite - Datenbankverwaltung*]

ULCreateNotificationQueue-Methode

Erstellt eine Ereignisbenachrichtigungs-Warteschlange für diese Verbindung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **name** Der Name der neuen Warteschlange.

- **parameters** Derzeit nicht verwendet. Auf NULL setzen.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Der Bereich von Warteschlangennamen gilt für die jeweilige Verbindung, daher können verschiedene Verbindungen Warteschlangen mit demselben Namen erstellen. Wenn eine Ereignisbenachrichtigung gesendet wird, empfangen alle Warteschlangen in der Datenbank mit einem übereinstimmenden Namen eine (separate Instanz der) Benachrichtigung. Namen reagieren nicht auf Groß- und Kleinschreibung. Beim Aufrufen der ULRegisterForEvent-Methode wird für jede Verbindung eine Standard-Warteschlange erstellt, falls keine Warteschlange angegeben ist. Dieser Abfruf schlägt mit einem Fehler fehl, wenn der Name bereits vorhanden oder nicht gültig ist.

ULDeclareEvent-Methode

Deklariert ein Ereignis, für das anschließend eine Registrierung erfolgen kann und das ausgelöst werden kann.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **event_name** Der Name für ein neues benutzerdefiniertes Ereignis.

Rückgabe

TRUE, wenn das Ereignis erfolgreich deklariert wurde, und FALSE, wenn der Name bereits verwendet wird oder ungültig ist.

Bemerkungen

In UltraLite gibt es vordefinierte Systemereignisse, die von Vorgängen in der Datenbank oder in der Umgebung ausgelöst werden. Diese Funktion deklariert benutzerdefinierte Ereignisse. Benutzerdefinierte Ereignisse werden mit der ULTriggerEvent-Methode ausgelöst. Der Ereignisname muss eindeutig sein. Namen berücksichtigen nicht die Groß- und Kleinschreibung.

Siehe auch

- ULTriggerEvent-Methode [UltraLite Embedded SQL] auf Seite 260

ULDeleteAllRows-Methode

Löscht alle Zeilen aus einer Tabelle.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **number** Die ID der Tabelle, die gekürzt werden soll.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE. Dies ist z.B. der Fall, wenn die Tabelle nicht geöffnet ist oder wenn ein SQL-Fehler aufgetreten ist.

Bemerkungen

In einigen Anwendungen kann es nützlich sein, alle Zeilen aus einer Tabelle zu löschen, bevor eine neue Datenmenge in die Tabelle geladen wird. Wenn die StopSync-Eigenschaft für diese Verbindung eingestellt ist, werden die gelöschten Zeilen nicht synchronisiert.

> **Hinweis**
> Nicht festgeschriebene Einfügungen von anderen Verbindungen werden nicht gelöscht. Nicht festgeschriebene Löschungen von anderen Verbindungen werden ebenfalls nicht gelöscht, wenn die andere Verbindung nach dem Aufrufen der DeleteAllRows-Methode ein Rollback durchführt.

Wenn diese Tabelle ohne Index geöffnet wurde, wird sie als schreibgeschützt angesehen und es können keine Daten gelöscht werden.

ULDestroyNotificationQueue-Methode

Zerstört die angegebene Ereignisbenachrichtigungs-Warteschlange.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **name** Der Name der zu zerstörenden Warteschlange.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Eine Warnung wird signalisiert, wenn ungelesene Benachrichtigungen in der Warteschlange verbleiben. Ungelesene Benachrichtigungen werden verworfen. Die Standard-Ereigniswarteschlange einer Verbindung, falls erstellt, wird vernichtet, wenn die Verbindung geschlossen wird.

ULEnableAesDBEncryption-Methode

Aktiviert die AES Datenbankverschlüsselung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den initialisierten SQLCA-Bereich

Bemerkungen

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der ULInitDatabaseManager-Methode aufrufen.

> **Hinweis**
> Wenn diese Methode aufgerufen wird, werden die Verschlüsselungsroutinen in die Anwendung einbezogen. Der Umfang des Anwendungscodes nimmt dadurch zu.

ULEnableAesFipsDBEncryption-Methode

Aktiviert die FIPS 140-2-zertifizierte AES-Datenbankverschlüsselung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den initialisierten SQLCA-Bereich

Bemerkungen

> **Hinweis**
> Wenn diese Methode aufgerufen wird, werden die entsprechenden Routinen in die Anwendung einbezogen. Die Größe des Anwendungscodes nimmt dadurch zu.

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der Synchronize-Methode aufrufen. Wenn Sie versuchen, eine Synchronisation durchzuführen, ohne zuvor den Synchronisationstyp mit einem Aufruf zu aktivieren, wird der SQLE_METHOD_CANNOT_BE_CALLED-Fehler ausgegeben.

Hinweis
Erforderliche getrennt lizenzierbare Komponenten.

FIPS-zertifizierte Verschlüsselung erfordert eine separate Lizenz. Alle Technologien für starke Verschlüsselungen unterliegen Exportbestimmungen.

Siehe „Getrennt lizenzierbare Komponenten“ [*SQL Anywhere 16 - Einführung*].

Siehe auch

- ULEnableAesDBEncryption-Methode [UltraLite Embedded SQL] auf Seite 239

ULEnableHttpSynchronization-Methode

Aktiviert die HTTP-Synchronisation

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Bemerkungen

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der Synchronize-Methode aufrufen. Wenn Sie versuchen, eine Synchronisation durchzuführen, ohne zuvor den Synchronisationstyp mit einem Aufruf zu aktivieren, wird der SQLE_METHOD_CANNOT_BE_CALLED-Fehler ausgegeben.

ULEnableRsaE2ee-Methode

Aktiviert die RSA-Ende-zu-Ende-Verschlüsselung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

ULEnableRsaFipsE2ee-Methode

Aktiviert die FIPS 140-2-zertifizierte RSA-Ende-zu-Ende-Verschlüsselung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

ULEnableRsaFipsSyncEncryption-Methode

Aktiviert die FIPS-Verschlüsselung für SSL- oder TLS-Datenströme.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Bemerkungen

Dies ist erforderlich, wenn ein Datenstrom auf TLS oder HTTPS gesetzt wird.

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der Synchronize-Methode aufrufen. Wenn Sie versuchen, eine Synchronisation durchzuführen, ohne zuvor den Synchronisationstyp mit einem Aufruf zu aktivieren, wird der SQLE_METHOD_CANNOT_BE_CALLED-Fehler ausgegeben.

Siehe auch

- ULEnableRsaSyncEncryption-Methode [UltraLite Embedded SQL] auf Seite 241

ULEnableRsaSyncEncryption-Methode

Aktiviert die RSA-Verschlüsselung für SSL- oder TLS-Datenströme.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Bemerkungen

Dies ist erforderlich, wenn ein Datenstrom auf TLS oder HTTPS gesetzt wird.

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der Synchronize-Methode aufrufen. Wenn Sie versuchen, eine Synchronisation durchzuführen, ohne zuvor den Synchronisationstyp mit einem Aufruf zu aktivieren, wird der SQLE_METHOD_CANNOT_BE_CALLED-Fehler ausgegeben.

Siehe auch

- ULEnableRsaFipsSyncEncryption-Methode [UltraLite Embedded SQL] auf Seite 241

ULEnableTcpipSynchronization-Methode

Aktiviert die TCP/IP-Synchronisation.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Bemerkungen

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der Synchronize-Methode aufrufen. Wenn Sie versuchen, eine Synchronisation durchzuführen, ohne zuvor den Synchronisationstyp mit einem Aufruf zu aktivieren, wird der SQLE_METHOD_CANNOT_BE_CALLED-Fehler ausgegeben.

ULEnableZlibSyncCompression-Methode

Aktiviert die ZLIB-Kompression für einen Synchronisationsdatenstrom

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den initialisierten SQLCA-Bereich

Bemerkungen

Diese Methode kann in C++-API- und Embedded SQL-Anwendungen verwendet werden. Sie müssen diese Methode vor der Synchronize-Methode aufrufen. Wenn Sie versuchen, eine Synchronisation durchzuführen, ohne zuvor den Synchronisationstyp mit einem Aufruf zu aktivieren, wird der SQLE_METHOD_CANNOT_BE_CALLED-Fehler ausgegeben.

ULErrorInfoInitFromSqlca-Methode

Kopiert die Fehlerinformationen aus dem SQLCA-Bereich auf das ul_error_info-Objekt.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

- **errinf** Das ul_error_info-Objekt.

ULErrorInfoParameterAt-Methode

Ruft einen Fehlerparameter nach Ordinalzahl ab.

Syntax

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

Parameter

- **errinf** Das ul_error_info-Objekt.
- **parmNo** Die auf 1 basierende Parameter-Ordinalzahl.
- **buffer** Der Puffer zur Aufnahme der Parameterzeichenfolge
- **bufferSize** Die Größe des Puffers.

Rückgabe

Die in Byte angegebene erforderliche Größe zum Speichern des Parameters oder 0, wenn die Ordinalzahl nicht gültig ist. Wenn der Rückgabewert den bufferSize-Wert überschreitet, wurde der Parameter gekürzt.

ULErrorInfoParameterCount-Methode

Ruft die Anzahl der Fehlerparameter ab.

Syntax

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

Parameter

- **errinf** Das ul_error_info-Objekt.

Rückgabe

Die Anzahl der Fehlerparameter.

ULErrorInfoString-Methode

Ruft eine Kurzbeschreibung des Fehlers ab.

Syntax

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

Parameter

- **errinf** Das ul_error_info-Objekt.
- **buffer** Der Puffer, der die Fehlerbeschreibung aufnehmen soll
- **bufferSize** Die Größe des Puffers in Byte.

Rückgabe

Die in Byte ausgedrückte Größe des zum Speichern der Zeichenfolge erforderlichen Puffers. Wenn der Rückgabewert den len-Wert überschreitet, wurde die Zeichenfolge gekürzt.

ULErrorInfoURL-Methode

Ruft eine URL zur Dokumentationsseite für diesen Fehler ab.

Syntax

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

Parameter

- **errinf** Das ul_error_info-Objekt.
- **buffer** Der Puffer, der die URL aufnehmen soll
- **bufferSize** Die Größe des Puffers in Byte.
- **reserved** Reserviert für spätere Verwendung.

Rückgabe

Die in Byte ausgedrückte Größe des zum Speichern der URL erforderlichen Puffers. Wenn der Rückgabewert den len-Wert überschreitet, wurde die URL gekürzt.

ULGetDatabaseID-Methode

Ruft die aktuelle Datenbank-ID ab, die für global autoincrement verwendet wird.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Rückgabe

Der Wert, der durch den letzten Aufruf der SetDatabaseID-Methode festgelegt wurde, oder UL_INVALID_DATABASE_ID, wenn die ID nie festgelegt wurde.

ULGetDatabaseProperty-Methode

Ruft den Wert einer Datenbankeigenschaft ab.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **id** Der Bezeichner für die Datenbankeigenschaft
- **dst** Ein Zeichen-Array zum Speichern des Eigenschaftswerts
- **buffer_size** Die Größe des Zeichen-Arrays dst
- **null_indicator** Ein Indikator, der anzeigt, dass der Datenbankparameter NULL ist

ULGetErrorParameter-Methode

Fehlerparameter werden über eine Parameter-Ordinalzahl abgerufen.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **parm_num** Die Ordinalzahl des Parameters.
- **buffer** Ein Zeiger auf einen Puffer, der den Fehlerparameter enthält.
- **size** Die Größe des Puffers in Byte.

Rückgabe

Diese Methode gibt die Anzahl der Zeichen zurück, die in den angegebenen Puffer kopiert wurden.

Siehe auch

- ULGetErrorParameterCount-Methode [UltraLite Embedded SQL] auf Seite 246

ULGetErrorParameterCount-Methode

Ruft die Anzahl der Fehlerparameter ab.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Rückgabe

Die Anzahl der Fehlerparameter. Wenn das Ergebnis nicht Null ist, können Werte von 1 bis zu diesem Ergebnis verwendet werden, um die ULGetErrorParameter-Methode aufzurufen, damit der entsprechende Fehlerparameterwert abgerufen wird.

Siehe auch

- ULGetErrorParameter-Methode [UltraLite Embedded SQL] auf Seite 245

ULGetIdentity-Methode

Ruft den @identity-Wert ab.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Rückgabe

Der letzte eingefügte Wert in eine Autoincrement- oder Global-Autoincrement-Spalte.

ULGetLastDownloadTime-Methode

Fragt den letzten Zeitpunkt ab, zu dem eine bestimmte Publikation heruntergeladen wurde

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **pub_name** Eine Zeichenfolge, die den Namen der Publikation enthält, für die die Zeit des letzten Downloads abgerufen wird.
- **value** Ein Zeiger auf die DECL_DATETIME-Struktur, die gefüllt werden soll. Der Wert 1. Januar 1990 bedeutet beispielsweise, dass die Publikation noch nicht synchronisiert wurde.

Rückgabe

TRUE, wenn der Wert erfolgreich mit dem letzten Downloadzeitpunkt der durch den pub_name-Wert angegebenen Publikation gefüllt wurde, ansonsten FALSE.

Bemerkungen

Der folgende Aufruf füllt die dt-Struktur mit dem Datum und der Uhrzeit des letzten Downloads der Publikation UL_PUB_PUB1:

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

ULGetNotification-Methode

Liest eine Ereignisbenachrichtigung.

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
)
```

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **queue_name** Die zu lesende Warteschlange oder NULL für die Standard-Verbindungswarteschlange.

- **event_name_buf** Ein Puffer für die Aufnahme des Ereignisnamens.
- **event_name_buf_len** Die Größe des Puffers in Byte.
- **wait_ms** Die abzuwartende Zeitspanne (Blockierung, in Millisekunden) vor der Rückgabe.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Dieser Aufruf wird blockiert, bis eine Benachrichtigung empfangen wird oder die angegebene Wartezeit abgelaufen ist. Übergeben Sie UL_READ_WAIT_INFINITE für den wait_ms-Parameter, um einen unbegrenzten Wartezeitraum festzulegen. Um den Wartezustand zu beenden, senden Sie eine weitere Benachrichtigung an die angegebene Warteschlange oder verwenden Sie die ULCancelGetNotification-Methode. Verwenden Sie nach dem Lesen einer Benachrichtigung die ULGetNotificationParameter-Methode, um zusätzliche Parameter nach Namen abzurufen.

Siehe auch

- ULCancelGetNotification-Methode [UltraLite Embedded SQL] auf Seite 233
- ULGetNotificationParameter-Methode [UltraLite Embedded SQL] auf Seite 248

ULGetNotificationParameter-Methode

Ruft einen Parameter für die Ereignisbenachrichtigung ab, die gerade von der ULGetNotification-Methode gelesen wurde.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **queue_name** Zu lesende Warteschlange oder NULL für Standard-Verbindungswarteschlange.
- **parameter_name** Der Name des Parameters, der gefunden werden soll (oder "*").
- **value_buf** Ein Puffer für die Aufnahme des Parameterwerts.
- **value_buf_len** Die Größe des Puffers in Byte.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Es sind nur die Parameter aus der zuletzt gelesenen Benachrichtigung in der angegebenen Warteschlange verfügbar. Parameter werden nach Namen abgerufen. Mit dem Parameternamen "*" wird die komplette Parameterzeichenfolge abgerufen.

ULGetSyncResult-Methode

Ruft das Ergebnis der letzten Synchronisation ab.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **sync_result** Ein Zeiger auf die ul_sync_result-Struktur, die die Synchronisationsergebnisse enthält

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Siehe auch

- ul_sync_result-Struktur [UltraLite C- und Embedded SQL-Datentypen] auf Seite 99

ULGlobalAutoincUsage-Methode

Ruft den Prozentwert der Standardwerte ab, die in allen Spalten verwendet werden, die global autoincrement-Standardwerte haben.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Rückgabe

Der Prozentwert des Global-Autoincrement-Werts, der vom Zähler verwendet wird.

Bemerkungen

Wenn die Datenbank mehr als eine Spalte mit diesem Standardwert enthält, wird dieser Wert für alle Spalten berechnet und das Maximum wird zurückgegeben. Der Rückgabewert 99 weist z.B. darauf hin, dass zumindest für eine der Spalten sehr wenige Standardwerte übrig sind.

Siehe auch

- ULSetDatabaseID-Methode [UltraLite Embedded SQL] auf Seite 255

ULGrantConnectTo-Methode

Erteilt einer vorhandenen Benutzer-ID mit dem betreffenden Kennwort Zugriff auf eine UltraLite-Datenbank.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **uid** Ein Zeichen-Array, das die Benutzer-ID enthält.
- **pwd** Ein Zeichen-Array, das das Kennwort für die Benutzer-ID enthält.

Bemerkungen

Diese Methode aktualisiert das Kennwort für einen vorhandenen Benutzer, wenn Sie eine vorhandene Benutzer-ID angeben.

Siehe auch

- ULRevokeConnectFrom-Methode [UltraLite Embedded SQL] auf Seite 253

ULInitSyncInfo-Methode

Initialisiert die Synchronisationsinformationsstruktur.

Syntax

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

Parameter

- **info** Eine Synchronisationsstruktur.

ULIsSynchronizeMessage-Methode

Überprüft eine Nachricht, um festzustellen, ob es sich um eine Synchronisationsnachricht vom MobiLink-Provider für ActiveSync handelt, sodass Code für die Bearbeitung der betreffenden Nachricht aufgerufen werden kann

Syntax

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

Bemerkungen

Wenn die Verarbeitung einer Synchronisationsnachricht abgeschlossen ist, muss die ULSignalSyncIsComplete-Methode aufgerufen werden.

Sie müssen einen Aufruf dieser Methode in die WindowProc-Funktion Ihrer Anwendung einbeziehen. Dies gilt für Windows Mobile für ActiveSync.

Der folgende Codeausschnitt veranschaulicht, wie die ULIsSynchronizeMessage-Methode verwendet wird, um eine Synchronisationsnachricht zu bearbeiten:

```
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;
}
```

Siehe auch

- ULSignalSyncIsComplete-Methode [UltraLite Embedded SQL] auf Seite 257

ULLibraryVersion-Methode

Gibt die Versionsnummer der UltraLite-Laufzeitbibliothek zurück.

Syntax

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

Rückgabe

Die Versionsnummer der UltraLite-Laufzeitbibliothek.

ULRSALibraryVersion-Methode

Gibt die Versionsnummer der RSA-Verschlüsselungsbibliothek zurück.

Syntax

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

Rückgabe

Die Versionsnummer der RSA-Verschlüsselungsbibliothek.

ULRegisterForEvent-Methode

Registriert eine Warteschlange, um Benachrichtigungen für ein Ereignis zu empfangen, oder macht diese Registrierung rückgängig.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **event_name** Das system- oder benutzerdefinierte Ereignis, für das eine Registrierung erfolgen soll.
- **object_name** Das Objekt, für das das Ereignis gilt, z.B. ein Tabellenname.
- **queue_name** Der Name der Verbindungswarteschlange. NULL steht für die Standard-Verbindungswarteschlange.
- **register_not_unreg** TRUE, um die Warteschlange zu registrieren, FALSE, um die Registrierung aufzuheben.

Rückgabe

TRUE, wenn die Registrierung erfolgreich war, und FALSE, wenn die Warteschlange oder das Ereignis nicht vorhanden ist.

Bemerkungen

Wenn kein Warteschlangenname geliefert wird, wird die Standard-Verbindungswarteschlange angenommen und ggf. erstellt. Bei bestimmten Systemereignissen können Sie den Namen eines Objekts angeben, für das das Ereignis gilt. Das TableModified-Ereignis z.B. kann den Tabellennamen spezifizieren. Im Gegensatz zur ULSendNotification-Methode empfängt nur die jeweilige registrierte Warteschlange Benachrichtigungen des Ereignisses. Andere Warteschlangen mit demselben Namen auf anderen Verbindungen erhalten keine Benachrichtigungen, es sei denn, sie sind ebenfalls explizit registriert.

Die vordefinierten Systemereignisse sind:

- **TableModified** Wird ausgelöst, wenn Zeilen in einer Tabelle eingefügt, aktualisiert oder gelöscht werden. Eine Benachrichtigung wird pro Anforderung gesendet, unbeschadet der Anzahl der durch die Anforderung betroffenen Zeilen. Der Parameter object_name gibt die zu überwachende Tabelle an. Der Wert "*" steht für alle Tabellen in der Datenbank. Dieses Ereignis hat einen Parameter namens "table_name", dessen Wert der Name der geänderten Tabelle ist.

- **Commit** Wird ausgelöst, nachdem ein Festschreibvorgang abgeschlossen ist. Dieses Ereignis hat keine Parameter.

- **SyncComplete** Wird ausgelöst, nachdem die Synchronisation abgeschlossen ist. Dieses Ereignis hat keine Parameter.

ULResetLastDownloadTime-Methode

Diese Methode setzt die Zeit des letzten Downloads einer Publikation zurück, sodass die Anwendung bereits heruntergeladene Daten erneut synchronisiert.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich

- **pub_list** Eine Zeichenfolge mit einer kommagetrennten Liste von Publikationen, die zurückgesetzt werden sollen. Eine leere Zeichenfolge ordnet alle Tabellen zu, außer den als "no sync" markierten. Eine Zeichenfolge, die nur ein Sternchen ("*") enthält, bedeutet alle Publikationen. Einige Tabellen sind in keiner Publikation referenziert und werden daher nicht einbezogen, wenn die pub-list-Zeichenfolge "*" ist.

Bemerkungen

Der folgende Methodenaufruf setzt die Zeit des letzten Downloads für alle Tabellen zurück:

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

ULRevokeConnectFrom-Methode

Entzieht einer Benutzer-ID die Zugriffsberechtigung auf eine UltraLite-Datenbank

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

- **uid** Ein Zeichen-Array, das die Benutzer-ID speichert, die vom Zugriff auf die Datenbank ausgeschlossen werden soll.

ULRollbackPartialDownload-Methode

Setzt die Änderungen einer fehlgeschlagenen Synchronisation zurück.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Bemerkungen

Wenn während der Download-Phase einer Synchronisation ein Verbindungsfehler auftritt, kann UltraLite die heruntergeladenen Änderungen anwenden, sodass die Anwendung die Synchronisation an der Stelle wieder aufnehmen kann, an der sie unterbrochen wurde. Wenn die Download-Änderungen nicht benötigt werden (z.B. weil eine Wiederaufnahme des Downloads vom Benutzer oder der Anwendung an dieser Stelle nicht gewünscht wird), setzt die ULRollbackPartialDownload-Methode die fehlgeschlagene Download-Transaktion zurück.

ULSendNotification-Methode

Sendet eine Nachricht an alle Warteschlangen, die mit dem angegebenen Namen übereinstimmen.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **queue_name** Der Name der Verbindungswarteschlange. NULL bedeutet die Standard-Verbindungswarteschlange.
- **event_name** Vom System oder vom Benutzer definiertes Ereignis, für das eine Registrierung erfolgen soll.
- **parameters** Derzeit nicht verwendet. Auf NULL setzen.

Rückgabe

Anzahl der gesendeten Benachrichtigungen (die Anzahl der übereinstimmenden Warteschlangen).

Bemerkungen

Dies umfasst jede Warteschlange in der aktuellen Verbindung. Dieser Aufruf bewirkt keine Blockierung. Verwenden Sie den speziellen Warteschlangennamen "*", um Benachrichtigungen an alle Warteschlangen zu senden. Der angegebene Ereignisname muss nicht mit einem vom System oder einem Benutzer definierten Ereignis übereinstimmen. Er wird einfach weitergereicht, um die Benachrichtigung beim Lesen zu identifizieren und ist nur für den Sender und Empfänger von Bedeutung.

Der Wert parameters wird in Form einer durch Semikolons getrennten Optionsliste aus Name=Wert-Paaren übergeben. Nachdem die Nachricht gelesen wurde, werden die Parameterwerte mit der ULGetNotificationParameter-Methode gelesen.

Siehe auch

- ULGetNotificationParameter-Methode [UltraLite Embedded SQL] auf Seite 248

ULSetDatabaseID-Methode

Legt die Datenbank-Identifizierungsnummer fest

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **value** Eine positive Ganzzahl, die eine bestimmte Datenbank in einer Replikations- oder Synchronisationsumgebung eindeutig kennzeichnet.

Siehe auch

- ULGlobalAutoincUsage-Methode [UltraLite Embedded SQL] auf Seite 249

ULSetDatabaseOptionString-Methode

Legt eine Datenbankoption mit einem Zeichenfolgenwert fest

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **id** Der Bezeichner für die Datenbankoption, die festgelegt werden soll

- **value** Der Wert der Datenbankoption.

ULSetDatabaseOptionULong-Methode

Legt eine nummerische Datenbankoption fest.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **id** Der Bezeichner für die Datenbankoption, die festgelegt werden soll
- **value** Der Wert der Datenbankoption.

ULSetErrorCallback-Methode

Legt den Callback fest, der bei einem Fehler aufzurufen ist.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **callback** Die Callback-Funktion.
- **user_data** An den Callback übergebene Benutzerkontextinformationen.
- **buffer** Ein vom Benutzer eingegebener Puffer, der die Fehlerparameter enthält, wenn der Callback aufgerufen wird.
- **len** Die Größe des Puffers in Byte.

Siehe auch

- „Fehlerbehandlung“ auf Seite 22

ULSetSyncInfo-Methode

Erstellt ein Synchronisationsprofil unter Verwendung des angegebenen Namens basierend auf der angegebenen ul_sync_info-Struktur.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **profile_name** Der Name des Synchronisationsprofils.
- **sync_info** Ein Zeiger auf die ul_sync_info-Struktur, die die Synchronisationsparameter enthält.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Das Synchronisationsprofil ersetzt alle vorherigen Profile mit demselben Namen. Das benannte Profil wird durch die Angabe eines Nullzeigers für die Struktur gelöscht.

ULSetSynchronizationCallback-Methode

Legt den Callback fest, der während einer Synchronisation aufgerufen werden soll.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **callback** Der Callback.
- **user_data** An den Callback übergebene Benutzerkontextinformationen.

ULSignalSyncIsComplete-Methode

Gibt an, dass die Verarbeitung einer Synchronisationsnachricht abgeschlossen ist.

Syntax

```
public ul_ret_void ULSignalSyncIsComplete()
```

Bemerkungen

Anwendungen, die beim ActiveSync-Provider registriert sind, müssen diese Methode in WNDPROC aufrufen, wenn die Verarbeitung einer Synchronisationsnachricht abgeschlossen ist.

ULStartSynchronizationDelete-Methode

Legt START SYNCHRONIZATION DELETE für diese Verbindung fest.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

ULStaticFini-Methode

Führt die Finalisierung der UltraLite-Laufzeitumgebung für Embedded SQL-Anwendungen durch.

Syntax

```
public void ULStaticFini()
```

Bemerkungen

Diese Methode darf nur einmal pro Anwendung aufgerufen werden und danach dürfen keine anderen UltraLite-Methoden mehr aufgerufen werden.

ULStaticInit-Methode

Führt die Initialisierung der UltraLite-Laufzeitumgebung für Embedded SQL-Anwendung durch.

Syntax

```
public void ULStaticInit()
```

Bemerkungen

Diese Methode darf nur einmal pro Anwendung aufgerufen werden und erst danach dürfen andere UltraLite-Methoden aufgerufen werden.

ULStopSynchronizationDelete-Methode

Legt STOP SYNCHRONIZATION DELETE für diese Verbindung fest.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

ULSynchronize-Methode

Initiiert die Synchronisation in einer UltraLite-Anwendung

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **info** Ein Zeiger auf die ul_sync_info-Struktur, die die Synchronisationsparameter enthält.

Bemerkungen

Bei der TCP/IP- oder HTTP-Synchronisation startet die ULSynchronize-Methode die Synchronisation. Fehler während der Synchronisation, die nicht vom Skript handle_error behandelt werden, werden als SQL-Fehler gemeldet. Anwendungsprogramme müssen den SQLCODE-Rückgabewert dieser Methode testen.

Das folgende Beispiel zeigt die Datenbanksynchronisation:

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

ULSynchronizeFromProfile-Methode

Synchronisiert die Datenbank unter Verwendung der angegebenen Parameter profile und merge.

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
)
```

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **profile_name** Der Name des zu synchronisierenden Profils.
- **merge_parms** Merge-Parameter für die Synchronisation.
- **observer** Observer-Callback, an den Statusaktualisierungen gesendet werden.
- **user_data** An den Callback übergebene Benutzerkontextdaten.

Bemerkungen

Diese Methode ist identisch mit dem Ausführen der SYNCHRONIZE-Anweisung.

Siehe auch

- „SYNCHRONIZE-Anweisung [UltraLite]“ [*UltraLite - Datenbankverwaltung*]

ULTriggerEvent-Methode

Löst ein benutzerdefiniertes Ereignis aus (und sendet eine Benachrichtigung an alle registrierten Warteschlangen).

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich
- **event_name** Vom System oder vom Benutzer definiertes Ereignis, für das eine Registrierung erfolgen soll.
- **parameters** Derzeit nicht verwendet. Auf NULL setzen.

Rückgabe

Die Anzahl der gesendeten Ereignisbenachrichtigungen.

Bemerkungen

Der Wert parameters wird in Form einer durch Semikolons getrennten Optionsliste aus Name=Wert-Paaren übergeben. Nachdem die Nachricht gelesen wurde, werden die Parameterwerte mit der ULGetNotificationParameter-Methode gelesen.

Siehe auch

- ULGetNotificationParameter-Methode [UltraLite Embedded SQL] auf Seite 248

ULTruncateTable-Methode

Kürzt die Tabelle und aktiviert vorübergehend die STOP SYNCHRONIZATION DELETE-Anweisung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **number** Die ID der Tabelle, die gekürzt werden soll.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

ULValidateDatabase-Methode

Validiert die Datenbank in dieser Verbindung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **start_parms** Der Parameter zum Starten der Datenbank.
- **table_id** Die ID einer bestimmten zu validierenden Tabelle.
- **flags** Parameter, die die Art der Validierung steuern.

- **callback_fn** Funktion, die die Informationen über den Verarbeitungsfortschritt der Validierung erhält.
- **user_data** Benutzerdaten, die über den Callback an den Aufrufer zurückzusenden sind.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Je nach den an die Routine übergebenen Parametern können die gespeicherten Basisdaten bzw. die Indizes validiert werden. Um während der Validierung Informationen zu erhalten, implementieren Sie eine Callback-Funktion und übergeben die Adresse an diese Routine. Um die Validierung auf eine bestimmte Tabelle zu beschränken, übergeben Sie den Tabellennamen oder die ID als letzten Parameter.

Der flags-Parameter ist eine Kombination aus folgenden Werten:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Siehe auch

- ULVF_TABLE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 105
- ULVF_INDEX-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104
- ULVF_DATABASE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_EXPRESS-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_FULL_VALIDATE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104

ULValidateDatabaseTableName-Methode

Validiert die Datenbank in dieser Verbindung.

Syntax

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

Parameter

- **sqlca** Ein Zeiger auf den SQLCA-Bereich.
- **start_parms** Der Parameter zum Starten der Datenbank.
- **table_name** Der Name einer bestimmten zu validierenden Tabelle.
- **flags** Parameter, die die Art der Validierung steuern.
- **callback_fn** Funktion, die die Informationen über den Verarbeitungsfortschritt der Validierung erhält.
- **user_data** Benutzerdaten, die über den Callback an den Aufrufer zurückzusenden sind.

Rückgabe

TRUE bei Erfolg, ansonsten FALSE.

Bemerkungen

Je nach den an die Routine übergebenen Parametern können die gespeicherten Basisdaten bzw. die Indizes validiert werden. Um während der Validierung Informationen zu erhalten, implementieren Sie eine Callback-Funktion und übergeben die Adresse an diese Routine. Um die Validierung auf eine bestimmte Tabelle zu beschränken, übergeben Sie den Tabellennamen oder die ID als letzten Parameter.

Der flags-Parameter ist eine Kombination aus folgenden Werten:

- ULVF_TABLE
- ULVF_INDEX
- ULVF_DATABASE
- ULVF_EXPRESS
- ULVF_FULL_VALIDATE

Siehe auch

- ULVF_TABLE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 105
- ULVF_INDEX-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104
- ULVF_DATABASE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_EXPRESS-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 103
- ULVF_FULL_VALIDATE-Variable [UltraLite C- und Embedded SQL-Datentypen] auf Seite 104

ul_database_option_id-Enumeration

Gibt mögliche Datenbankoptionen an, die Benutzer festlegen können.

Syntax

```
public enum ul_database_option_id
```


Mitglieder

Mitgliedsname	Beschreibung
ul_option_global_database_id	Die globale Datenbank-ID wird mit einer langen Ganzzahl ohne Vorzeichen festgelegt.
ul_option_ml_remote_id	Die entfernte ID wird mithilfe einer Zeichenfolge festgelegt.
ul_option_commit_flush_timeout	Der Timeout für das Bereinigen einer Datenbankfestschreibung wird als Ganzzahl festgelegt, die einen Zeitschwellenwert in Millisekunden darstellt.
ul_option_commit_flush_count	Der Zähler für das Bereinigen bei einem Datenbankfestschreiben wird als Ganzzahl festgelegt, die einen Zahlschwellenwert darstellt.
ul_option_isolation_level	Die Isolationsstufe für die Verbindung wird als Zeichenfolge festgelegt. (read_committed/read_uncommitted)
ul_option_cache_allocation	Hierdurch wird die Größe des Datenbankdateicaches geändert. Der Wert ist eine Ganzzahl zwischen 0 und 100, die die Menge an Cache darstellt, der dem Bereich zwischen der minimalen und der maximalen Größe zugewiesen ist.

Bemerkungen

Diese Datenbankoptionen werden zusammen mit der ULConnection.SetDatabaseOption-Methode verwendet.

Siehe auch

- ULConnection.SetDatabaseOption-Methode [UltraLite C++] auf Seite 129

ul_database_property_id-Enumeration

Gibt mögliche Datenbankeigenschaften zurück, die die Benutzer abrufen können.

Syntax

```
public enum ul_database_property_id
```

Mitglieder

Mitgliedsname	Beschreibung
ul_property_date_format	Datumsformat. (date_format)

Mitgliedsname	Beschreibung
ul_property_date_order	Datumsreihenfolge. (date_order)
ul_property_nearest_century	Nächstes Jahrhundert. (nearest_century)
ul_property_precision	Gesamtstellenzahl. (precision)
ul_property_scale	Dezimalstellenzahl. (scale)
ul_property_time_format	Zeitformat. (time_format)
ul_property_timestamp_format	Zeitstempelformat. (timestamp_format)
ul_property_timestamp_increment	Zeitstempel-Inkrementierungsstufe. (timestamp_increment)
ul_property_name	Name. (Name)
ul_property_file	DAtei. (File)
ul_property_encryption	Verschlüsselung. (Encryption)
ul_property_global_database_id	Globale Datenbank-ID. (global_database_id)
ul_property_ml_remote_id	Entfernte ID. (ml_remote_id)

Mitgliedsname	Beschreibung
ul_property_char_set	Zeichensatz. (CharSet)
ul_property_collation	Kollationssequenz. (Collation)
ul_property_page_size	Seitengröße. (PageSize)
ul_property_case_sensitive	Berücksichtigung von Groß-/Kleinschreibung. (CaseSensitive)
ul_property_conn_count	Verbindungsanzahl. (ConnCount)
ul_property_max_hash_size	Standardmäßige Maximal-Index-Hashgröße. (MaxHashSize)
ul_property_checksum_level	Datenbankprüfsummenebene. (ChecksumLevel)
ul_property_checkpoint_count	Datenbank-Checkpointanzahl. (CheckpointCount)
ul_property_commit_flush_timeout	Timeout für das Bereinigen einer Datenbankfestschreibung. (commit_flush_timeout)
ul_property_commit_flush_count	Zahl für das Bereinigen beim Datenbank-Festschreiben. (commit_flush_count)
ul_property_isolation_level	Isolationsstufe der Verbindung. (isolation_level)
ul_property_time-stamp_with_time_zone_format	Zeitstempel mit Zeitzonenformat. (timestamp_with_time_zone_format)

Mitgliedsname	Beschreibung
ul_property_cache_allocation	Die aktuelle Größe des Datenbankdateicache als Prozentsatz des Bereichs zwischen dem Minimum und dem Maximum.

Bemerkungen

Diese Eigenschaften werden zusammen mit der ULConnection.GetDatabaseProperty-Methode verwendet.

Siehe auch

- ULConnection.GetDatabaseProperty-Methode [UltraLite C++] auf Seite 119

ml_file_transfer_info-Struktur

Eine Struktur mit den Parametern für den Datei-Upload/Download.

Syntax

```
public typedef struct ml_file_transfer_info
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
auth_parms	const char *	Stellt Authentifizierungsparametern in MobiLink-Ereignissen Parameter bereit. Weitere Hinweise finden Sie unter „Synchronisationsparameter Additional Parameters“ [*UltraLite - Datenbankverwaltung*].
auth_status	asa_uint16	Stellt Authentifizierungsparametern in MobiLink-Ereignissen Parameter bereit. Weitere Hinweise finden Sie unter „Synchronisationsparameter Additional Parameters“ [*UltraLite - Datenbankverwaltung*].
auth_value	asa_uint32	Gibt Auskunft über die Ergebnisse eines benutzerdefinierten Benutzerauthentifizierungsskripts in MobiLink. Der MobiLink-Server gibt diese Information an den Client weiter. Weitere Hinweise finden Sie unter „Authentication Value-Synchronisationsparameter“ [*UltraLite - Datenbankverwaltung*].

Mitgliedsname	Typ	Beschreibung
enable_resume	bool	Wenn dies auf TRUE gesetzt ist, nimmt die MLFileDownload-Methode einen vorangegangenen Download wieder auf, der durch einen Verbindungsfehlers unterbrochen oder vom Benutzer abgebrochen wurde. Wenn die Datei auf dem Server neuer ist als die lokale Teildatei, wird die Teildatei entfernt und die neue Version in ihrer Gesamtheit heruntergeladen. Der Standardwert ist TRUE.
error	mlft_stream_error	Enthält Informationen über jeden auftretenden Fehler
file_auth_code	asa_uint16	Enthält den Rückgabecode des optionalen Skripts authenticate_file_transfer auf dem Server
filename	const char *	Der Name der Datei, die vom Server übertragen werden soll, auf dem MobiLink ausgeführt wird. MobiLink durchsucht zuerst das Benutzername-Unterverzeichnis und anschließend das Stammverzeichnis. Weitere Hinweise finden Sie unter „mlsrv16-Option -ftr “ [*MobiLink - Serveradministration*].
local_filename	const char *	Der lokale Name der heruntergeladenen Datei. Wenn dieser Parameter leer ist, wird der Wert in "filename" verwendet.
local_path	const char *	Der lokale Pfad, in dem die heruntergeladene Datei gespeichert werden soll. Wenn dieser Parameter leer ist (Standard), wird die heruntergeladene Datei im aktuellen Verzeichnis gespeichert. Unter Windows Mobile wird die Datei im Stammverzeichnis (\) des Geräts gespeichert, wenn dest_path leer ist. Auf dem PC wird die Datei im aktuellen Benutzerverzeichnis gespeichert, wenn dest_path leer ist.

Mitgliedsname	Typ	Beschreibung
num_auth_parms	asa_uint8	Die Anzahl der Authentifizierungsparameter, die an die Authentifizierungsparameter in MobiLink-Ereignissen übergeben werden. Weitere Hinweise finden Sie unter „Synchronisationsparameter Number of Authentication Parameters“ [*UltraLite - Datenbankverwaltung*].
observer	ml_file_transfer_observer_fn	Eine Callback-Funktion kann über das Feld "observer" bereitgestellt werden, um den Fortschritt des Datei-Downloads zu beobachten. Weitere Hinweise finden Sie in der nachfolgenden Beschreibung der Callback-Funktion.
password	const char *	Das Kennwort für den MobiLink-Benutzernamen
remote_key	const char *	Der entfernte MobiLink-Schlüssel.
stream	const char *	Das Protokoll kann eines der folgenden sein: TCPIP, TLS, HTTP oder HTTPS. Dieses Feld ist erforderlich. Weitere Hinweise finden Sie unter „Synchronisationsparameter Stream Type“ [*UltraLite - Datenbankverwaltung*].
stream_parms	const char *	Die Protokolloptionen für einen bestimmten Datenstrom. Weitere Hinweise finden Sie unter „UltraLite-Netzwerkprotokolloptionen für dbmlsync“ [*UltraLite - Datenbankverwaltung*].
transferred_file	asa_uint16	Gibt 1 zurück, wenn die Datei erfolgreich übertragen wurde, und 0, falls ein Fehler auftritt. Ein Fehler tritt auf, falls die Datei bereits aktuell ist, wenn MLFileUpload aufgerufen wird. In diesem Fall gibt die Funktion TRUE anstelle von FALSE zurück.
user_data	void *	Die anwendungsspezifischen Informationen, die der Synchronisations-Beobachtungsfunktion bereitgestellt werden. Weitere Hinweise finden Sie unter „Synchronisationsparameter User Data“ [*UltraLite - Datenbankverwaltung*].

Mitgliedsname	Typ	Beschreibung
username	const char *	Der MobiLink-Benutzername Dieses Feld ist erforderlich.
version	const char *	Die MobiLink-Skriptversion. Dieses Feld ist erforderlich.

ml_file_transfer_status-Struktur

Eine Struktur, die den Status/die Informationen über den Verarbeitungsfortschritt während des Datei-Uploads/Downloads enthält.

Syntax

```
public typedef struct ml_file_transfer_status
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
bytes_transferred	asa_uint64	Gibt an, wie viel von der Datei bislang heruntergeladen wurde, einschließlich der Werte aus früheren Synchronisationen, falls der Download wieder aufgenommen wurde.
file_size	asa_uint64	Gibt die Gesamtgröße der heruntergeladenen Datei in Byte an.
flags	asa_uint16	Stellt zusätzliche Informationen bereit. Der MLFT_STATUS_FLAG_IS_BLOCKING-Wert wird gesetzt, wenn die MLFileDownload-Methode bei einem Netzwerkaufruf blockiert und der Download-Status sich seit dem letzten Aufruf der Beobachtungsmethode nicht geändert hat.
info	ml_file_transfer_info *	Zeigt auf das Informationsobjekt, dass an die MLFileDownload-Methode übergeben wurde. Sie können über diesen Zeiger auf den Parameter user_data zugreifen.
resumed_at_size	asa_uint64	Wird zusammen mit der Download-Wiederaufnahme verwendet und gibt an, an welchem Punkt der aktuelle Download wieder aufgenommen wurde

Mitgliedsname	Typ	Beschreibung
stop	asa_uint8	Setzen Sie dies auf TRUE, um den aktuellen Download abzubrechen. Sie können den Download durch einen nachfolgenden Aufruf der MLFileDownload-Methode wieder aufnehmen, jedoch nur, wenn der enable_resume-Parameter gesetzt wurde.

mlft_stream_error-Struktur

Eine Struktur, die Informationen zum Status/Verarbeitungsfortschritt während des Datei-Uploads bzw. -Downloads enthält.

Syntax

```
public typedef struct mlft_stream_error
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
error_string	char	Die lokalisierte Beschreibung für den system_error_code-Wert, sofern aus dem System verfügbar, oder zusätzliche Informationen für den stream_error_code-Wert.
stream_error_code	ss_error_code	Der spezifische Datenstromfehler. Eine Liste der möglichen Werte finden Sie unter der ss_error_code-Enumeration in der Header-Datei *%SQLANY16%\SDK\Include\sserror.h*.
system_error_code	asa_int32	Ein systemspezifischer Fehlercode.

Siehe auch

- „Kommunikationsfehlermeldungen des MobiLink-Servers - nach Fehlercodes sortiert“ [*Fehlermeldungen*]

mlft_stream_error_w-Struktur

Eine Struktur, die Informationen zum Status/Verarbeitungsfortschritt während des Datei-Uploads bzw. -Downloads enthält.

Syntax

```
public typedef struct mlft_stream_error_w
```

Mitglieder

Mitgliedsname	Typ	Beschreibung
error_string	wchar_t	Die lokalisierte Beschreibung für den system_error_code-Wert, sofern aus dem System verfügbar, oder zusätzliche Informationen für den stream_error_code-Wert.
stream_error_code	ss_error_code	Der spezifische Datenstromfehler. Eine Liste der möglichen Werte finden Sie unter der ss_error_code-Enumeration in der Header-Datei *%SQLANY12%\SDK\Include\sserror.h*.
system_error_code	asa_int32	Ein systemspezifischer Fehlercode.

Bemerkungen

Hinweis
Dieser Strukturprototyp wird intern verwendet, wenn Sie die mlft_stream_error-Struktur referenzieren und den UNICODE-Makro auf Win32-Plattformen festlegen (#define). In der Regel referenzieren Sie diese Struktur nicht direkt bei der Erstellung einer UltraLite-Anwendung.

Siehe auch

- mlft_stream_error-Struktur [UltraLite Embedded SQL] auf Seite 271
- „Kommunikationsfehlermeldungen des MobiLink-Servers - nach Fehlercodes sortiert“ [*Fehlermeldungen*]

MLFT_STATUS_FLAG_IS_BLOCKING-Variable

Definiert eine Bitgruppe im ml_file_transfer_status.flags-Feld, um anzuzeigen, dass die Übertragung blockiert wurde, weil eine Antwort des MobiLink-Servers erwartet wird.

Syntax

```
#define MLFT_STATUS_FLAG_IS_BLOCKING
```

Bemerkungen

Identische Nachrichten zum Dateiübertragungsfortschritt werden in regelmäßigen Abständen generiert, während dies der Fall ist.

Index

Symbole

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Z