Tutorial: Build an Application Using the C++ Component

Contents

About this tutorial

This tutorial guides you through the process of building a simple UltraLite C++ Component application. The application is built for Windows operating systems, and runs at a command prompt. This tutorial uses Visual Studio to edit the C++ file. You can also use any C++ development environment or text editor.

1 Introduction

Timing

The tutorial takes about 30 minutes if you copy and paste the code. If you enter the code yourself, it takes significantly longer.

Competencies and experience

This tutorial assumes:

- you are familiar with the C++ programming language
- you have a C++ compiler installed on your computer
- you know how to create an UltraLite schema using the UltraLite Schema Painter

Goals

The goals for the tutorial are to gain competence and familiarity with the process of developing an UltraLite C++ Component application.

2 Lesson 1: Connect to the database

In the first procedure, you create a database schema. You then write, compile, and run a C++ application that creates a database using the schema you have created.

❖ To create a database schema

1. Create a directory to hold the files you create in this tutorial.

   The remainder of this tutorial assumes that this directory is *c:\tutorial\cpp*. If you create a directory with a different name, use that directory instead of *c:\tutorial\cpp* throughout the tutorial.

2. Using the UltraLite Schema Painter, create a database schema in your new directory with the following characteristics.

   Schema file name: **tutcustomer.usm**

   Table name: **customer**

   Columns in customer:

Column Name	Data Type (Size)	Column allows NULL values?	Default value
id	integer	No	autoincrement
fname	char(15)	No	None
lname	char(20)	No	None
city	char(20)	Yes	None
phone	char(12)	Yes	555-1234

   Primary key: ascending **id**

❖ To connect to an UltraLite database

1. In Microsoft Visual C++, choose File ➤ New.

2. On the Files tab, choose C++ Source File.

3. Save the file as *customer.cpp* in your tutorial directory.

4. Import the UltraLite libraries and use the UltraLite namespace.

   Copy the code below into *customer.cpp*.

```
#include "uliface.h"
#include <stdio.h>
#include <tchar.h>
#include <assert.h>
using namespace UltraLite;
#define MAX_NAME_LEN 100
ULSqlca Sqlca;
```

5. Define connection parameters to connect to the database. In this example, the parameters are the location of the database and schema files.

In the following code, these locations are hard coded. In a real application, the locations would be specified at runtime. In addition, these connection parameters are sufficient only for connections in the development environment; additional parameters are needed for the application to run on a Windows CE device.

Copy the code below into *customer.cpp*.

```
static ul_char const * ConnectionParms =
  UL_TEXT( "UID=DBA;PWD=SQL" )
  UL_TEXT( ";DBF=tutcustomer.udb" )
  UL_TEXT( ";schema_file=tutcustomer.usm" );
```

6. Define a method for error handling.

   UltraLite provides a callback mechanism to notify the application of errors.

   This is a sample callback function.

```
enum { ERROR_MSG_LEN = 140,
        ERROR_CONTEXT_LEN = 20,
        ERROR_CALLBACK_BUF_LEN = 80 };

ul_error_action UL_GENNED_FN_MOD MyErrorCallBack(
   SQLCA *    sqlca,
   ul_void *   user_data,
   ul_char *   buffer )
{
     ul_error_action action;
    (void) user_data;
    (void) buffer;

    switch( sqlca->sqlcode ){
          // The following errors are used for flow control,
          // and we don't want to report them here.
          case SQLE_NOTFOUND:
          case SQLE_ULTRALITE_DATABASE_NOT_FOUND:
              // Suppress these warnings.
              action = UL_ERROR_ACTION_DEFAULT;
              break;
          case SQLE_CANNOT_ACCESS_SCHEMA_FILE:
              _tprintf( _TEXT(
              "Error %ld; UltraLite schema file not found\n" ),
              sqlca->sqlcode );
              action = UL_ERROR_ACTION_CANCEL;
              break;
          case SQLE_COMMUNICATIONS_ERROR:
              _tprintf( _TEXT(
              "Error %ld: Communications error\n" ),
               sqlca->sqlcode );
              action = UL_ERROR_ACTION_DEFAULT;
              break;
          default:
              _tprintf( _TEXT(
              "Error %ld: \n" ),
              sqlca->sqlcode );
              action = UL_ERROR_ACTION_DEFAULT;
              break;
    }
    return action;
}
```

In UltraLite, two errors are used to control application flow. The SQLE_ULTRALITE_DATABASE_NOT_FOUND error is signaled on the first connection attempt (when only the schema file is present), and is used to prompt the application to create a database from the schema file. The SQLE_NOTFOUND error marks the end of a loop over a result set.

7. Define a method to open a connection to a database.

   If the database file does not exist, a SQLException is thrown. The schema file is used to create a new database and establish a connection to it.

   If the database file exists, a connection is established.

```
Connection * open_conn( DatabaseManager * dm )
{
    Connection * conn;

    conn = dm->OpenConnection( Sqlca, ConnectionParms );
    if( conn == NULL ) {
          if( Sqlca.GetSQLCode() == SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
              // The database doesn’t exist yet --
              // create it using the schema file.
              conn = dm->CreateAndOpenDatabase( Sqlca, ConnectionParms );
                    _tprintf( _TEXT("Connected to a new database.\n") );
        }
    } else {
          _tprintf( _TEXT("Connected to an existing database.\n") );
    }
    return conn;
}
```

8. Implement the main() method.

   The main method carries out the following tasks.

- Instantiates a DatabaseManager object. All UltraLite objects are created from the DatabaseManager object.
- Registers the error handling function.
- Opens a connection to the database.
- Closes the connection and shuts down the database manager.

```
int main()
{
    DatabaseManager * dm;
    Connection * conn;
    ul_char      buffer[ERROR_CALLBACK_BUF_LEN];


    Sqlca.Initialize();
    ULRegisterErrorCallback(
              Sqlca.GetCA(),
              &MyErrorCallBack,
              UL_NULL,
              buffer,
              ERROR_CALLBACK_BUF_LEN );
    dm = ULInitDatabaseManager( Sqlca );
    if( dm == UL_NULL ){
          // You may have mismatched UNICODE vs. ANSI runtimes.
          Sqlca.Finalize();
          return 1;
    }
```

```
        conn = open_conn( dm );
        if( conn == UL_NULL ){
              dm->Shutdown( Sqlca );
              Sqlca.Finalize();
              return 1;
        }

        conn->Release();
        dm->Shutdown( Sqlca );
        Sqlca.Finalize();
        return 0;
    }
```

9. Compile and link the Customer class.

   The method you use to compile the class depends on your compiler. The following instructions are for the Microsoft Visual C++ command line compiler using a makefile.

   - From a command prompt, browse to your tutorial directory.
   - Create a makefile named *makefile*.
   - In the makefile, add directories to your include path as follows.

     ```
     IncludeFolders= \
     /I"$(ASANY9)\h"
     ```

   - In the makefile, add directories to your libraries path as follows.

     ```
     LibraryFolders= \
     /LIBPATH:"$(ASANY9)\ultralite\win32\386\lib"
     ```

   - In the makefile, add libraries to your linker command line options as follows.

     ```
     Libraries= \
     ulimp.lib
     ```

     The UltraLite runtime library, *ulimp.lib*, is an ASCII version of the library. If you choose the Unicode version, *ulimpw.lib*, you should add /DUNICODE to the compiler options.

   - In the makefile, set the following compiler options all on one line.

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

   - In the makefile, add an instruction for linking the application.

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

   - In the makefile, add an instruction for compiling the application.

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

   - Add an instruction to create a preprocessed version of the file. This step is included for debugging purposes.

     ```
     customer.i: customer.cpp
        cl $(CompileOptions) $(IncludeFolders) customer.cpp -P
     ```

   - Run the makefile as follows:

     ```
     nmake
     ```

     An executable named *customer.exe* is created.

10. Run the application.

    At the command prompt, enter **customer**.

3 Lesson 2: Insert data into the database

The following procedures demonstrate how to add data to a database.

❖ To add rows to your database

1. Add procedure below to *customer.cpp*, immediately before the main method.

   This procedure carries out the following tasks.

   - Opens the table using the `connection->OpenTable()` method. You must open a Table object to carry out operations on the table.
   - Obtains identifiers for the required columns of the table. The other columns in the table can accept NULL values or have a default value.
   - If the table is empty, adds two rows. To insert each row, the code changes to insert mode using the InsertBegin method, sets values for each required column, and executes an insert to add the rows to the database.

     The commit method is only required when you turn off autocommit. By default, autocommit is enabled but it may be disabled for better performance, or for multi-operation transactions.
   - If the table is not empty, reports the number of rows in the table.
   - Closes the Table object.
   - Returns a boolean indicating whether the operation was completed.

```
bool do_insert( Connection * conn )
{
    Table * table = conn->OpenTable( _TEXT("customer") );
    if( table == NULL ) {
          return false;
    }

    if( table->GetRowCount() == 0 ) {
          _tprintf( _TEXT("Inserting two rows.\n") );
          table->InsertBegin();
          table->Set( _TEXT("fname"), _TEXT("Penny") );
          table->Set( _TEXT("lname"), _TEXT("Stamp") );
          table->Insert();
        table->InsertBegin();
          table->Set( _TEXT("fname"), _TEXT("Gene") );
          table->Set( _TEXT("lname"), _TEXT("Poole") );
          table->Insert();

          conn->Commit();
    } else {
          _tprintf( _TEXT("The table has %lu rows\n"), table->GetRowCount() );
    }
    table->Release();
    return true;
}
```

2. Call the do_insert method you have created.

   Add the following line to the `main()` method, immediately after the call to open_conn.

   ```
   do_insert(conn);
   ```

3. Compile your application by running *nmake*.

4. Run your application by typing *customer* at the command prompt.

4 Lesson 3: Select the rows from the table

The following procedure retrieves rows from the table and prints them on the command line.

❖ To list the rows in the table

1. Add the method below to *customer.cpp*. This method carries out the following tasks.
   - Opens the Table object.
   - Retrieves the column identifiers.
   - Sets the current position before the first row of the table.

     Any operations on a table are carried out at the current position. The position may be before the first row, on one of the rows of the table, or after the last row. By default, as in this case, the rows are ordered by their primary key value (id). To order rows in a different way, you can add an index to an UltraLite database and open a table using that index.
   - For each row, the id and name are written out. The loop carries on until the Next method returns false, which occurs after the final row.
   - Closes the Table object.

   ```
   bool do_select( Connection * conn )
   {
       Table * table = conn->OpenTable( _TEXT("customer") );
       if( table == NULL ) {
             return false;
       }

       TableSchema * schema = table->GetSchema();
       if( schema == NULL ) {
             table->Release();
             return false;
       }
       ul_column_num id_cid    = schema->GetColumnID( _TEXT("id") );
       ul_column_num fname_cid = schema->GetColumnID( _TEXT("fname") );
       ul_column_num lname_cid = schema->GetColumnID( _TEXT("lname") );
       schema->Release();

       while( table->Next() ){
             char fname[ MAX_NAME_LEN ];
             char lname[ MAX_NAME_LEN ];
   ```

```
            table->Get( fname_cid ).GetString( fname, MAX_NAME_LEN );
              table->Get( lname_cid ).GetString( lname, MAX_NAME_LEN );
              _tprintf( "id=%d, name=%s %s\n", (int)table->Get( id_cid ),
                       fname, lname );
        }
        table->Release();
        return true;
    }
```

2. Add the following line to the `main()` method, immediately after the call to the insert method:

   ```
   do_select(conn);
   ```

3. Compile your application by running *nmake*.

4. Run your application by typing *customer* at the command prompt.

5 Lesson 4: Add synchronization to your application

This lesson synchronizes your application with a consolidated database running on your computer.

The following procedures add synchronization code to your application, start the MobiLink synchronization server, and run your application to synchronize.

> **Note**
> This lesson uses MobiLink synchronization, which is part of SQL Anywhere Studio. You must have SQL Anywhere Studio installed on your computer to carry out this lesson.

The UltraLite database you created in the previous lessons synchronizes with the UltraLite 9.0 Sample database. The UltraLite 9.0 sample database has a ULCustomer table whose columns match those in the customer table of your UltraLite database.

This lesson assumes that you are familiar with MobiLink synchronization.

❖ To add synchronization to your application

1. Add the method below to *customer.cpp*. This method carries out the following tasks.
   - Sets the synchronization stream to TCP/IP. Synchronization can also be carried out over HTTP, ActiveSync, or HTTPS.
   - Sets the script version. MobiLink synchronization is controlled by scripts stored in the consolidated database. The script version identifies which set of scripts to use.
   - Sets sendColumnNames to true so the MobiLink synchronization server can generate synchronization scripts automatically.
   - Sets the MobiLink user name. This value is used for authentication at the MobiLink synchronization server. It is distinct from the UltraLite database user ID, although in some applications you may wish to give them the same value.
   - Sets the download_only parameter to true. By default, MobiLink synchronization is two-way. This application uses download-only synchronization so that the rows in your table do not get uploaded to the sample database.

```
bool do_sync( Connection * conn )
{
  ul_synch_info    info;

  conn->InitSynchInfo( &info );
  info.stream = ULSocketStream();
  info.version = UL_TEXT( "ul_default" );
  info.user_name = UL_TEXT( "sample" );
  info.send_column_names = true;
  info.download_only = true;
  if( !conn->Synchronize( &info ) ) {
      return false;
  }
  return true;
}
```

2. Add the following line to the `main()` method, immediately after the call to the insert method and before the call to the select method.

   ```
   do_sync(conn);
   ```

3. Compile your application by running *nmake*.

❖ To synchronize your data

1. Start the MobiLink synchronization server.

   From a command prompt, run the following command.

   ```
   dbmlsrv9 -c "dsn=UltraLite 9.0 Sample" -v+ -zu+ -za
   ```

   The `-zu+` and `-za` command line options provide automatic addition of users and generation of synchronization scripts.

2. Run your application by typing *customer* at the command prompt.

   The MobiLink synchronization server window displays status messages indicating the synchronization progress. If synchronization is successful, the final message displays `Synchronization complete`.

6 Lesson 5: Deploy to a Windows CE device

The following procedure demonstrates how to deploy an UltraLite C++ Component application to a Windows CE device.

❖ To deploy to a Windows CE device

1. Ensure that your device is connected to your computer.

2. Start File Explorer on your device.

   Choose Start ➤ Programs ➤ File Explorer.

3. Create directories to hold the UltraLite runtime and application.
   - Navigate to the root of the device. The root may be named My Device or My Pocket PC.
   - Create a directory named *UltraLite*.

- Open the UltraLite directory and create subdirectories named *lib* and *CustDB*.

  \UltraLite\lib is the location for the UltraLite runtime files, and *\UltraLite\CustDB* is the location for the application.

4. Copy the UltraLite runtime files to the Windows CE device.

   You can now run the application on your Windows CE device. This completes the tutorial.