I Wish I Knew How To …

Program SQLite with Xojo API2

January 2021 Edition (2.0)

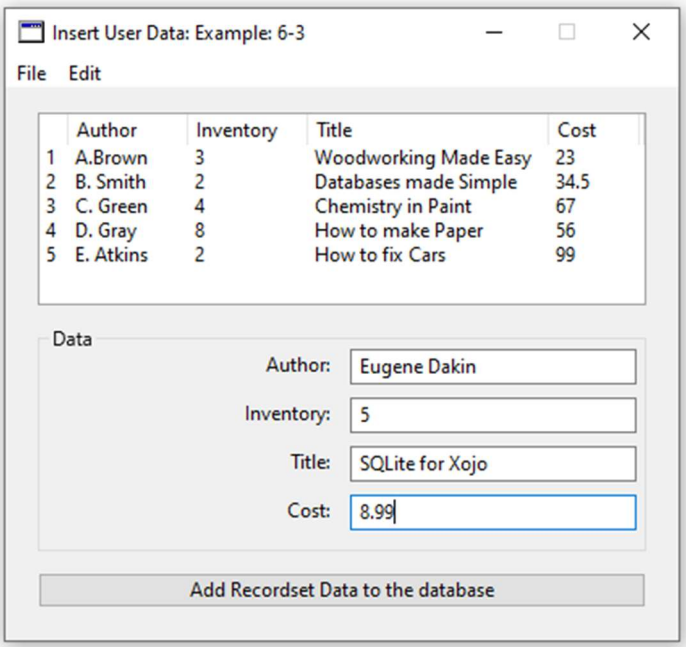
By Eugene Dakin

Table of Contents

Inserting RowSet Data

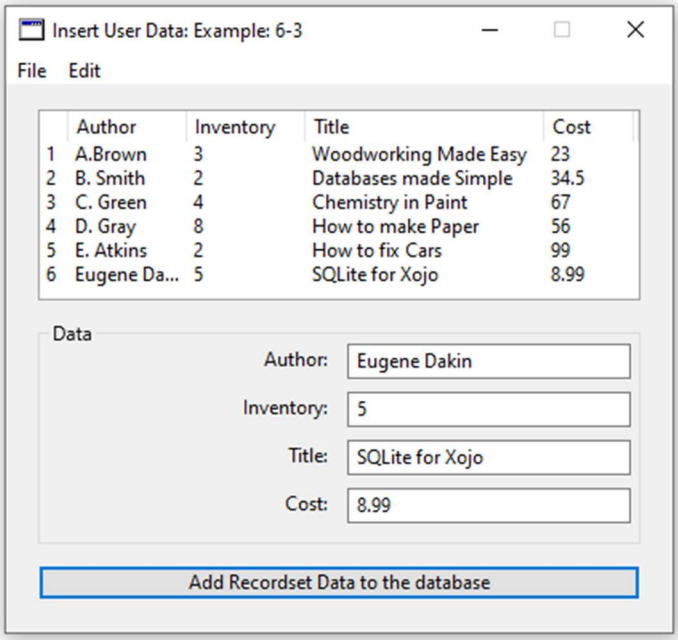
User-based data is entered into the database by modifying inserting text Column data into the SQL command. The following screenshot is of the program before the push button has been pressed. Data has been manually typed in for the Author, Inventory, Title, and Cost in the Text Columns. This example uses data addition with a RowSet instead of an SQL command.

Figure 65. Screengrab of Example 6-3 Prior to New Data



Data from the Insert.sqlite database is shown in the listbox and 5 entries are shown. There is data in the textColumns, such as Eugene Dakin for a new author, and the pushbutton has not yet been pressed. Once pressed, the following screen shot adds the data and updates the listbox from the database.

Figure 66. Example 6-3 with Data



When the program runs for the first time, the PushButton1.Open method is called. Within this method are the following calls to two other methods.

```
Sub Open() Handles Open
  Call ChooseDatabase
  Call RefreshListbox
End Sub
```

The ChooseDatabase method contains code for the user to choose the Database name and get the location of the database file. A global property is created called db, which is a SQLite database type.

Code 38. Example 6-3: ChooseDatabase Method

```
Public Sub ChooseDatabase()
  //Get the file name
  Var dlg as new OpenFileDialog
  dlg.Title = "Open *.sqlite File"
  f = dlg.ShowModal()

  //Connect to a SQLite database
```

```
  db = new SQLiteDatabase
  db.DatabaseFile = f

  If f <> Nil Then
    Try
      db.Connect
      //The connection was successful!
    Catch err as DatabaseException //It did not connect
      MessageDialog.Show "An error occured connecting to the database" + EndOfLine +
err.Message
      return
    End Try
  End If

  Exception err as OLEException //Let the user know of errors
    MessageDialog.Show err.Message
End Sub
```

A folderitem variable called f is created to hold the file name and to remember the directory where the file exists. An OpenFileDialog is added to prompt the user for the database filename and to get the location directory of the database. When the dialogue is open, the title at the top of the window shows 'Open *.sqlite File'. This provides a strong hint as to what action the user is to perform and the file extension name to open the Inventory SQLite database. A new instance of SQLite database variable called db is created. The filename and location of the file stored in the variable f is loaded into the databasefile.

If there is a database connection, meaning there are no errors and the database information was successfully read by the program, then tell the database connection was successful (db.Connect = True). If something caused an error, then let the user know about the error and also the error message. To make sure that all errors are shown, an Exception err is created to trap and show the error message.

Now that the database has been chosen, and the program has only asked the user once for the database location, the next step is to show the data in the listbox, or RefreshListbox.

Code 39. Example 6-3: RefreshListbox

```
Public Sub RefreshListbox()
  //Format the Listbox
  Listbox1.HasHeader = true //show the heading
  Listbox1.ColumnCount = 5 //make 5 columns
  Listbox1.ColumnWidths = "5%, 20%, 20%, 40%, 15%"
  Listbox1.InitialValue = "ID" + chr(9) + "Author" + chr(9) + "Inventory" + chr(9) + "Title" +
chr(9) + "Cost"

  //Get the data from the database
  Try
    Var rs as RowSet
    rs = db.SelectSQL("SELECT * FROM Library")
    If rs <> Nil Then
      For Each row as DatabaseRow in rs
        Listbox1.AddRow
        Listbox1.CellValueAt(Listbox1.LastAddedRowIndex, 0) =
Cstr(rs.Column("ID").IntegerValue)
        Listbox1.CellValueAt(Listbox1.LastAddedRowIndex, 1) = rs.Column("Author").StringValue
        Listbox1.CellValueAt(Listbox1.LastAddedRowIndex, 2) =
Cstr(rs.Column("Inventory").DoubleValue)
        Listbox1.CellValueAt(Listbox1.LastAddedRowIndex, 3) = rs.Column("Title").StringValue
        Listbox1.CellValueAt(Listbox1.LastAddedRowIndex, 4) =
Cstr(rs.Column("Cost").DoubleValue)
      Next
      rs.Close
    End If
    rs = Nil
  Catch error As DatabaseException
    MessageBox("Error: " + error.Message)
  End Try
End Sub
```

The listbox is formatted by showing the heading, making 5 columns, showing the column width in percentage values and adding the heading labels. Each label is separated by a tab that is

shown as chr(9). A RowSet variable is created with the variable name rs. A SelectSQL command is used and the results are to be placed in the rs variable. This is the step where SQL is implemented. The SELECT command means 'get' information from the database (as opposed to updating or adding new data). The wildcard character (*) means select everything in the database.

Once data has been retrieved from the database, the data then needs to be placed on the screen so the user can see the data. To prevent an error, the rs RowSet is checked to see if there is data (rs <> Nil). If there is data, then then each Column and its data is placed in the appropriate TextColumn. There is more than one record in this RowSet, and a do-until loop is used to start from the first RowSet and stop at the last RowSet. The IntegerValue 'ID' is place in the CellValueAt (0) of the new row (addrow). The column to the right has a value of 1, and contains the Author StringValue. Each Column from the database populates the listbox until the end of the database file has been reached. If there is no data in the RowSet, then a messagebox shows that there is no data and to gracefully prevent an error from happening. An Exception error bit of code has been added to capture any other errors.

Code 40. Pushbutton Update Code for Example 6-3

```
Sub Action() Handles Action
  Try
    //Insert data in the database
    Var row as new DatabaseRow
    row.Column("Author") = TFAuthor.Text
    row.Column("Inventory") = TFInventory.Text
    row.Column("Title")  = TFTitle.Text
    row.Column("Cost") = TFCost.Text

    db.AddRow("Library", row)

  Catch error As DatabaseException
    MessageBox("Error: " + error.Message)
  End Try

  call RefreshListbox
End Sub
```

Adding data to a database by a RowSet requires a few more lines of code and is a little easier to prevent mistakes. A new DatabaseRow variable is created called *row* and each column in the SQLite database is placed in quotation marks and the text (string) value is placed in the RowSet. Once all data has been entered in the RowSet then the *AddRow* adds a row to the table *Library*. If there is an error, error trapping shows some information to try and help the programmer track down the error. The listbox data is refreshed by calling the RefreshListbox method.

This example updates a SQLite database with a database record.

Index

The 'I Wish I Knew' series contains technical data and advice that makes sense and contains practical and numerous examples with explanations to allow you to ease into the steep programming curve. You can create Desktop SQLite database applications today!

This book "I Wish I Knew How to … Program SQLite with Xojo API2" delves into the mystery of programming a back-end database. SQLite is a zero-configuration, serverless, transactional, and self-contained database.

The book is written as a guide and reference to Xojo programmers who program Desktop Applications in Windows and Mac. There are no dynamic link libraries (dll), COM, or Active X parts to add. Although the report maker has been explained and can make reports, a workaround is to set the default print margin properties to zero.

There are more than 20 chapters and over 400 pages with greater than 80 example programs.

Examples include opening recordsets, finding data, sorting, updating data, counting records, printing a report, deleting records, creating new records, creating reports, and more. Many screenshots have been added to show the results of the code with an index to help find topics quickly.

This is one of many books at Xojo Library. This book can be purchased online at http://www.xdevlibrary.com/ where many great Xojo and Real Studio resources are available.

Happy programming!

Eugene

Eugene Dakin MBA, Ph.D., P.Chem., is an author of Xojo and Real Studio reference materials and has many years of experience in the programming industry.

ISBN: 978-1-927924-29-7