Homework #2

Complete By: Tuesday, June 30th @ start of class
Assignment: GUI app + Join queries
Policy: Individual work only, late work *is not* accepted
Submission: electronic via Blackboard

Reading

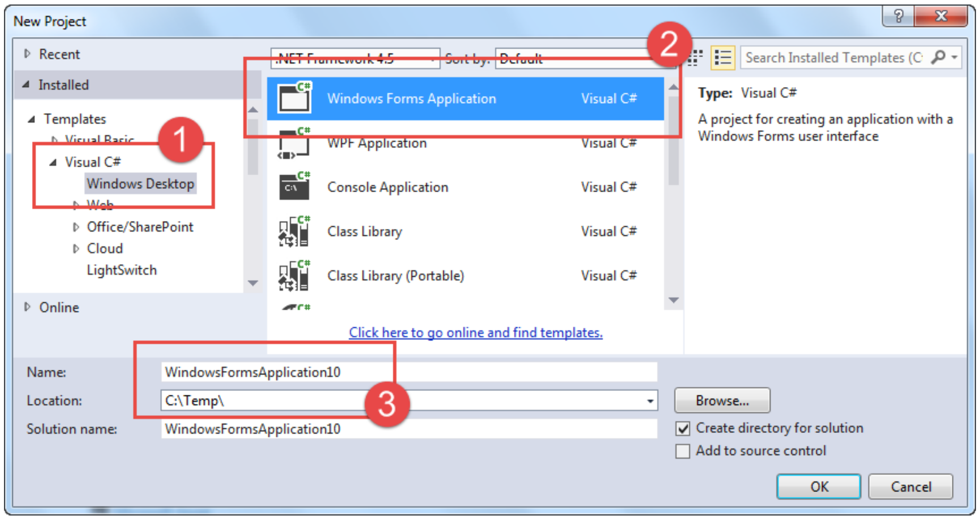
If needed, read about Join queries (online or in any available SQL textbook).

Database File

Download the **uiTunes** database from last Thursday's class: http://www.joehummel.net/cs480.html, open **Lectures**, open **06-25**, and download the "uiTunes.mdf" database file and associated log file to your desktop.

Event-driven (GUI) Programming in C#

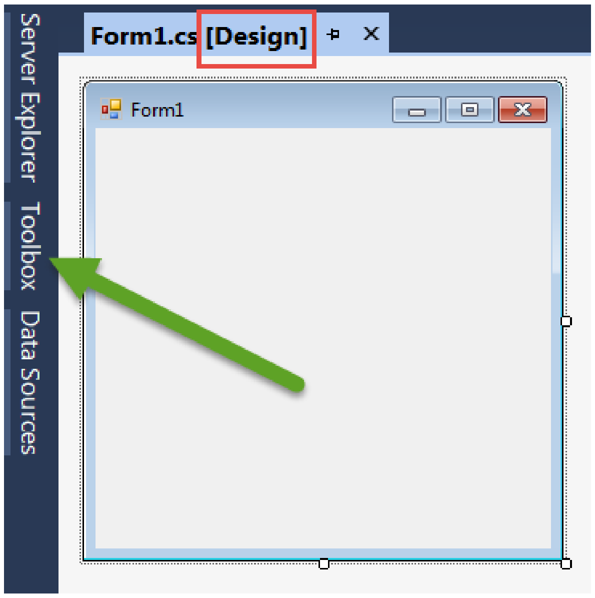
Let's create a GUI app in Visual Studio and C#. Startup Visual Studio, create a new project, and under "Templates" expand the options under **Visual C#** — see the screenshot shot below. Click on "Windows Desktop", and then to the right select "Windows Forms Application" (aka WinForms). Name the project, but more importantly, pay attention to the location where the project files are going to be saved (3):


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows Desktop
Web
Office/SharePoint
Cloud
LightSwitch
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Class Library (Portable) Visual C#
Type: Visual C#
A project for creating an application with a Windows Forms user interface
Click here to go online and find templates.
Browse...
Create directory for solution
Add to source control
OK
Cancel
1
2
3

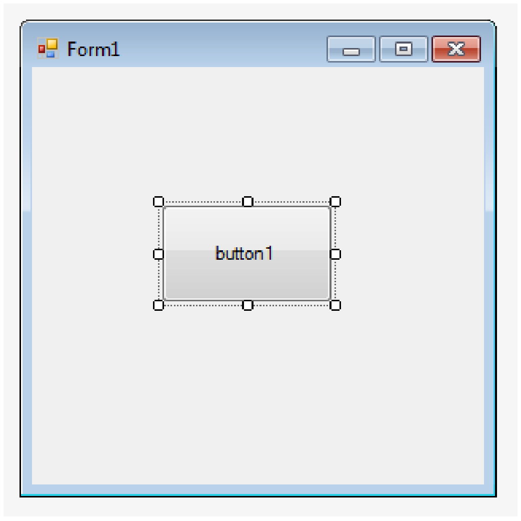
Name: WindowsFormsApplication10
Location: C:\Temp\
Solution name: WindowsFormsApplication10

Click OK to create the project. A new GUI-based app is created, and the main form (aka window) for your app is generated. Go ahead and run the program (F5). When you run, notice the main window is created and displayed, and that it automatically responds to many events: you can resize, you can grab the title bar and drag the window around, you can minimize and restore, and you can close. When you close the main window, the application halts. Go ahead and close the window.

Back in Visual Studio, let's do a little bit of event-driven programming. At this point you should be back in VS looking at the main window. Focus on the left margin, as shown on next page:

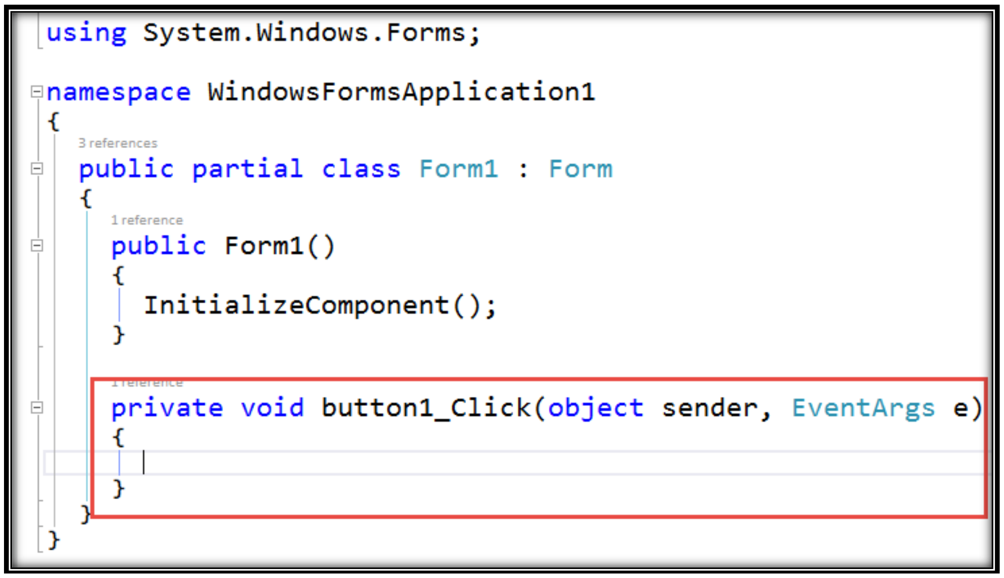


Notice the title bar above the main window — Visual Studio is telling you the form is being viewed in "Design" mode — this mode is for designing the user interface. Click on "Toolbox" on the left side (shown above), expand "Common Controls", click-and-hold on "Button", and drag to the right until the Toolbox closes. Then drop the button anywhere onto the form:



Run the application and click the button — notice that nothing happens. This makes sense, because we haven't programmed the button to respond yet to "being clicked".

Back in Visual Studio, double-click on the button in Design mode, and Visual Studio will reveal the "code-behind" window — this mode is for programming. Here's what you should be seeing (next page):



```
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }

        1 reference
        private void button1_Click(object sender, EventArgs e)
        {
            |
        }
    }
}
```

When you double-clicked on the button in Design mode, Visual Studio does 2 things:

1. Creates an event handler for responding to the click of this button
2. Hooks up this event handler so it is automatically called whenever the user clicks the button

Add the following line of code to the button's event handler method:

```
MessageBox.Show("click!");
```

Run, and click the button. A dialog box should open in response with the message "click!". This is the essence of event-driven programming.

Database programming using C# and ADO.NET object model

Let's modify our GUI app that when clicked, does the following:

1. Connects to the database
2. Executes SQL command to find song ID of "Timber"
3. Displays the returned id in a listbox
4. Experiment a bit more

Executing SQL amounts to forming the query as a string, sending it to the database server for execution, and then display the results.

Step 1 is to install the database and log file into the proper folder so when the program runs the database can be found. Minimize Visual Studio. Find the project folder where you saved the project; e.g. **C:\Temp**. Drill-down into the project folder (e.g. **WindowsFormApplication1**), and open the bin\Debug folder. Place a copy of the "uiTunes.mdf" database file (and its associated log file) in this folder. Repeat for the bin\Release folder.

Now back in Visual Studio, double-click on the button to modify the button's Click event handler. Code the button as follows…

1. Scroll to the top of the C# file, and add

```
using System.Data.SqlClient;
```

2. In the Click event handler, open a connection to the database, check the state to make sure it opened, and then immediately close the connection:

```
string          filename, connectionInfo;
SqlConnection   db;

filename = "uiTunes.mdf";
connectionInfo = String.Format(@"Data Source=(LocalDB)\v11.0;AttachDbFilena
me=|DataDirectory|\{0};Integrated Security=True;", filename);

db = new SqlConnection(connectionInfo);
db.Open();

MessageBox.Show(db.State.ToString());

db.Close();
```

3. Run (F5), click the button, and a dialog box should appear displaying "Open" --- this may take 10-15 seconds the first time you do this, since the DB engine "sqlservr.exe" has to startup. If the DB fails to open, double-check the **connectionInfo** string, one mistake in that string and the open will fail…

4. From the toolbox, drag-and-drop a Listbox onto the main form.

5. Comment out the call to MessageBox.Show. Between the call to Open and Close, add the following to execute our query against the database. Be careful with the syntax of the SQL string — both the " " and ' ' are necessary (think about why…):

```
//MessageBox.Show(dbConn.State.ToString());

string      sql, msg;
SqlCommand  cmd;
object      result;
```

```
sql = "select SongID from Songs where SongName = 'Timber;";
cmd = new SqlCommand();
cmd.Connection  = db;
cmd.CommandText = sql;

result = cmd.ExecuteScalar();

msg = String.Format("Song id:  {0}", result);

listBox1.Items.Add(msg);
```

6. Run (F5), click the button, and you should get the song id **2**.

The technology we are using is called ADO.NET ("Active Data Objects"). Let's experiment a bit more:

7. Add a textbox so the user can enter a song name. Now instead of always looking up 'Timber', we can lookup the user's favorite song. Change the sql string as follows:

   ```
   sql = string.Format("select SongID from Songs where SongName = '{0}';",
     this.textBox1.Text);

   MessageBox.Show(sql);
   ```

   string.Format is like printf --- the {0} is a placeholder for the first argument after the format string, which in this case is the string entered by the user. The MessageBox.Show(sql) will display the string so you'll see exactly what string.Format produces (a good debugging technique). Run and test. Trying looking up **Roar** and **Stairway To Heaven**. What happens if you lookup a song that doesn't exist? Can you come up with a better handling of song that are not found? [**Hint**: set a breakpoint by clicking in the left margin just before you display the result, run, and then lookup a song that doesn't exist. When the breakpoint triggers and Visual Studio stops, hover over the result variable to see what is returned when a song is not found.]

8. So far we have been retrieving one result from the database --- a song id. Retrieving a single value is accomplished using the ADO.NET **ExecuteScalar** function. If you want to retrieve multiple results, e.g. all the song names, you have to use a different technique in order to "fill" a container with the desired values. This container is called a **DataSet**, and models an in-memory database. Let's use this technique to display all the song names in the listbox. First, in design mode, drag-and-drop a 2nd button from the Toolbox to the main form. Double-click on the button to stub out an event handler. Scroll to the top of the C# file and add the following to gain access to the DataSet class:

   ```
   using System.Data;
   ```

   Copy-paste the code from the 1st button's event handler to the new handler you just added. Now change the SQL query string to retrieve all the song data from the songs table:

```
sql = string.Format("SELECT * FROM Songs ORDER BY SongName ASC;");
```

Then we execute the query, filling the dataset container. So comment out the call to ExecuteScalar, and replace with the following:

```
//result = cmd.ExecuteScalar();
//msg = String.Format("Song id:  {0}", result);
//listBox1.Items.Add(msg);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
adapter.Fill(ds);
```

At this point the dataset contains a temporary table with the data we asked for. The name of this table is **TABLE**. So now we grab the table from the dataset and let's output how many rows we retrieved:

```
DataTable dt = ds.Tables["TABLE"];

msg = String.Format("Retrieved {0} songs", dt.Rows.Count);

listBox1.Items.Add(msg);
```

There are 6 songs, so you should see the value **6** displayed. Finally, let's display the song names and the year each song was released. Each "row" in the table is an array of values, accessed by the database field name. So for each row in the table, we want to display the "SongName" and "YearRel" fields. Here's the code:

```
foreach (DataRow row in dt.Rows)
{
  msg = string.Format("{0}: {1}",
    row["SongName"].ToString(),
    row["YearRel"].ToString());

  listBox1.Items.Add(msg);
}
```

Run and test… That's it!

Joins, Joins, Joins

You won't submit the above GUI app, this was just an intro of what's to come in class and future homeworks. The following exercises are what you'll submit for this homework. Answer the following questions using the provided uiTunes database; test your SQL queries using your GUI app or a Visual Studio query window. Recall that an ER diagram for the database is also available on the course web page.

1. How many artists were involved in the song 'Timber'?

2. How many reviews have been posted by 'Pooja Sankar'?

3. What album names have been released by 'Pink Floyd'? Order by name in ascending order.

4. What genre(s) does 'Timber' fall into?

5. Names of all artists whose role has been "with" on a song.

Electronic Submission

Using Blackboard (http://uic.blackboard.com/), submit a text file or Word doc containing the answers to the Join exercises. Make sure your name and "HW 2" are at the top of your submission file. You may submit as many times as you want before the due date, but we grade the last version submitted.

Policy

Unless stated otherwise, all work submitted for grading *must* be done individually. While I encourage you to talk to your peers and learn from them, you cannot submit their work — whether partial or complete — as your own. The University's policy is described here: http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf . In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases may be submitted to the official student conduct process: http://www.uic.edu/depts/dos/studentconductprocess.shtml