W. P. CAREY SCHOOL of BUSINESS
ARIZONA STATE UNIVERSITY

CIS 345 – Business Information Systems Development II – Fall 2014

In-class Exercise 13: Office Automation - Using the Microsoft Excel Object Library

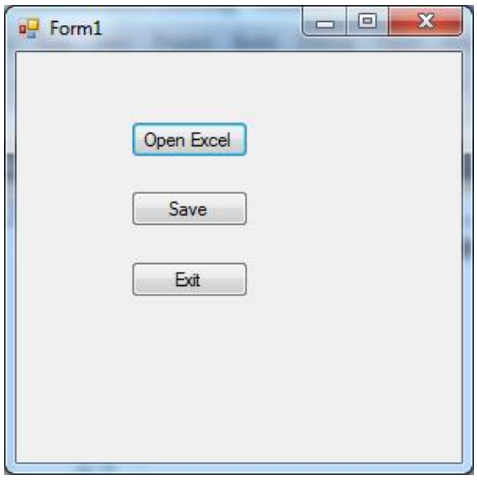
Due on Blackboard: Today, Monday, November 24 by 10 PM

Now that you have utilized objects and forms, the next step is to utilize object libraries that are supplied by software vendors. In this exercise, we will instantiate Excel objects by using the Microsoft Excel Object Library.

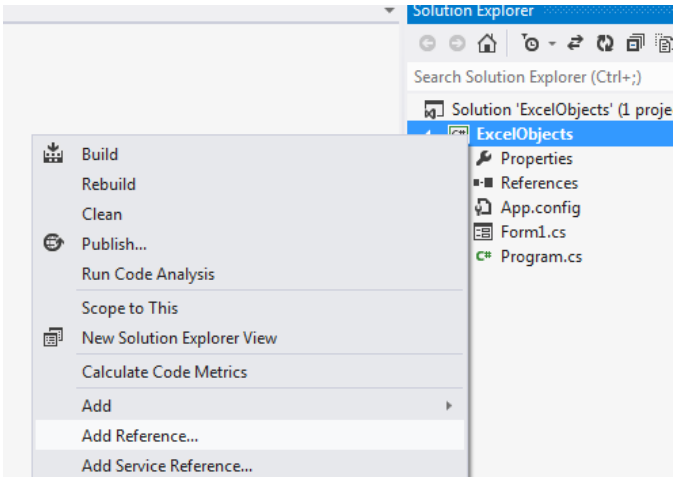
BOTH Excel and Visual Studio need to be installed for this exercise. If you do not have both Visual Studio AND Microsoft Office installed on the same operating system, you will be able to do this exercise from within Citrix, which does have both of those software products installed.

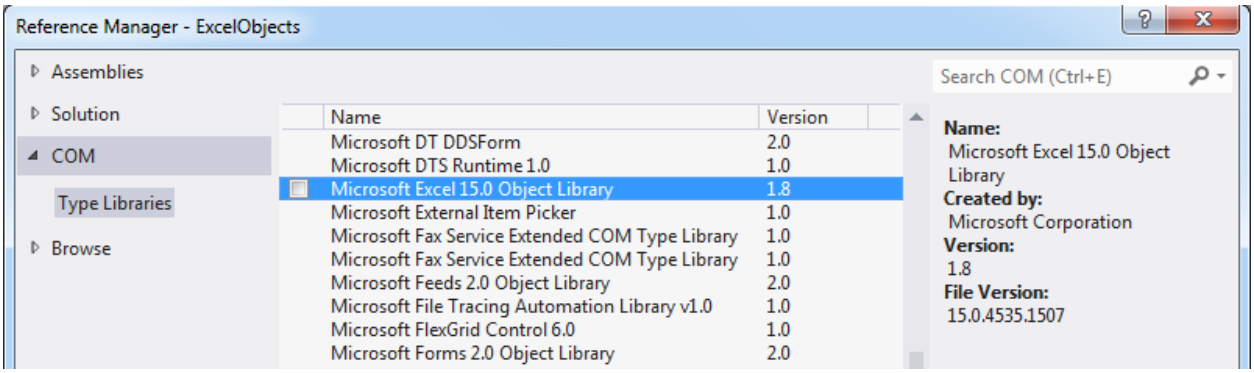
1) Create a Form Application and call your project, ExcelObjects

2) Create 3 buttons - Open Excel, Save File, Exit.



2) Add a reference to the Microsoft Excel Object Library by going into Add Reference and then "COM" (Component Object Model). The version number of the library will differ based on which version of Excel is installed. This COM object library provides the application interface to Microsoft Excel.





3) Within your ExcelObjects.cs partial class, add the following *using* statements:

```
using Excel = Microsoft.Office.Interop.Excel;
using System.Reflection;
```

The first *using* statement will allow you to refer to the namespace "Microsoft.Office.Interop.Excel" as merely "Excel." The reason we want to explicitly use the namespace name is that there will be some objects which share the same name across namespaces. For those cases, we need to explicitly refer to the namespace (but at the same time, we do not want to type the full namespace name over and over). The other using statement will allow you to use other needed classes.

4) Create an instance of Excel and work with the active sheet as shown below:

a) Declare ***instance*** variables that will hold references to Excel objects.

```
Excel.Application oXL          // Excel application object
Excel._Workbook oWB            // Excel Workbook object
Excel._Worksheet oSheet        // Excel Worksheet object
Excel._Worksheet oSheet2       // Excel Worksheet object
```

b) In your button Click event handler, create a new Excel Application. Assign it to your Application object.

```
oXL = new Excel.Application();
```

c) Set oXL's visible property to true.

d) Use the Excel Application's Add method to add a new workbook. It will return you an object reference - cast the object that is returned as "Excel._Workbook". Store the reference in your Workbook object reference, oWB.

```
oWB = (Excel._Workbook)(oXL.Workbooks.Add());
```

e) Obtain the active worksheet in the WorkBook from the ActiveSheet property. Cast the reference as "Excel._Worksheet" and store in your Worksheet reference, oSheet.

```
oSheet = (Excel._Worksheet) oWB.ActiveSheet;
```

f) Excel rows and columns start counting from row 1 and column 1. You can access an Excel cell by accessing the collection of cells in the format: "Cells[row#, column#]" just like an array.

Add table headers to row 1, columns 1 and 2

```
oSheet.Cells[1, 1] = "First Name";
oSheet.Cells[1, 2] = "Last Name";
```

This is the end of the first part of the activity – creating Excel sheets and manipulating cells. Add some more values to get practice in using Excel cells through C#.

5) *The next part of the activity is to create a new worksheet and work with it.*

a) Call the Add method of Worksheets. Cast the return value as “Excel._Worksheet”

```
oSheet2 = (Excel._Worksheet) oWB.Worksheets.Add();
```

b) Give the new Sheet a name.

```
oSheet2.Name = "Average Calculator";
```

c) Move the new sheet in front of the existing sheet you worked with in part 4. “Missing.Value” can be used in place of an argument that needs to be provided but one you do not have currently. This will move the sheet to the front.

```
oSheet2.Move(Missing.Value, oSheet);
```

d) Use a *for* loop to loop 5 times and assign a number to the cells in column 2. Start your loop counter from 1. Use the following code within your loop. This will place the value “i” in the cells in column 2.

```
oSheet2.Cells[i, 2] = i;
```

e) After the loop ends, store the text “Average” in Cell (row 6, column 1) in the same manner as in part (d).

f) Store an Excel formula to calculate average in Cell (row 6, column 2).

```
Cell location = "=AVERAGE(B1:B5)";
```

6) Create a Click event handler for your Save button. Within the event handler, use the SaveAs() method of the workbook object. Within the arguments, give it a filename. This file will be saved in My Documents by default.

```
oWB.SaveAs("MyExcel.xlsx");
```

7) Create a Click event handler for your Exit button. Within the event handler, call the Close() method of the workbook object and then the Quit() method of the Excel application object.

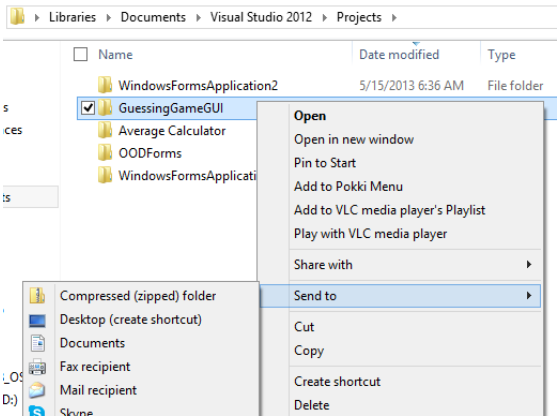
```
oWB.Close();
oXL.Quit();
```

Both these methods have the potential to generate exceptions. So, enclose them in a *try* block and catch a generic Exception. Within the *catch* block, call the Show method of MessageBox and tell the user that there was an error!

Submission Instructions

Submission should be made using a zip file that contains all of the Visual Studio C# project files. You will need to ***zip the entire project folder*** along with the .sln and .suo files. The folder will automatically contain the class source files as well as the executable file that is generated in \ProjectName\bin\Debug folder. Upload file to the Blackboard assignment drop box.

Zip the entire top-level folder by right-clicking the folder and selecting Send to | Compressed (zipped) folder.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Make sure you check the following. Your grade is dependent on all these criteria being met.

- You have included your name as a comment within your class.
    - e.g. " // In-class 13, Jane C. Smith, CIS 345, Tuesday 9:00 AM"
    - Your project is called ExcelObjects
- Zip filename is: FirstNameLastName_Inclass13.zip

Verify your zip file before you submit

- Check for actual class files being present in the folder before you zip it.
- Check your zip file size after zipping – if it is 1K, it likely contains only a shortcut.
- Uncompress your zip file before submitting and verify that files are present.
- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.
- Make sure you have submitted your file and not just saved a draft on Blackboard. A blue clock indicates a submission in progress, i.e. a draft, not a submission.