POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

www.dei.estg.ipleiria.pt

Systems Integration

C# Excel Interop

MICROSOFT INTEROP OBJECT

"Enhanced COM interop through C# dynamic type system, support for named and optional parameters, and for variance makes working with Microsoft Office and other Primary Interop Assemblies much easier"

by Joydip Kanjilal

Goals/Topics:

– Create, write, and read from excel files
– Manipulate excel files through COM Interop namespace

Requires:

– Visual Studio
– Microsoft Excel

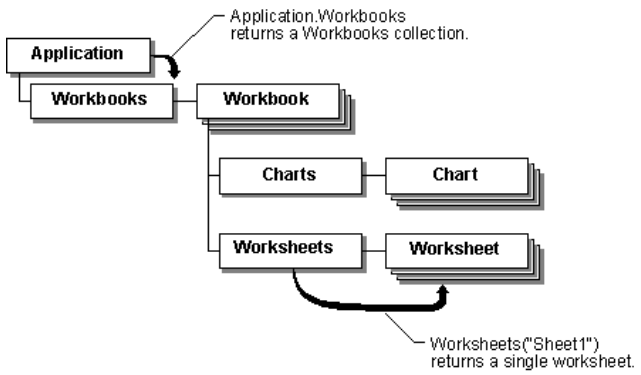

Figure 1 - Navigating down a Microsoft Excel object hierarchy using collections

Create a new project in Visual Studio (VS) using the **Class Library (.NET Framework)** template. Name the solution 'SolutionExcelAutomation' and name the project '**Excel-Lib'**.

Create a new class or rename the existing one (Class1.cs) to '**ExcelHandler'**. Figure 2 displays the structure of the project.

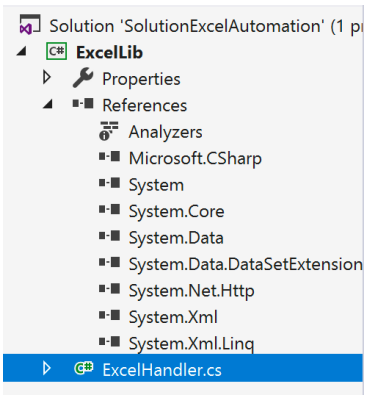

Figure 2 – Solution structure with the references to the objects being automated (Excel).

To use Excel operations in VS, the first and foremost thing to do is to include the Microsoft.Office Object Library reference to the project. Doing so will enable us to use the namespace **Microsoft.Office.Interop.Excel**. This namespace is widely referred in the project, hence I have assigned an alias name 'Excel'.

1. Prepare your project with Excel Object Data.

1.1. Add the Excel assembly as a reference (you add the reference **Microsoft.Office.Interop.Excel** by right-clicking on the project references > Add Reference > **COM section** (see Figure 2) and select the library ***Microsoft Excel 16.0 Object Library .***

Note: Depending on the version of Office installed the Excel Assembly may be called Excel 16.0 Object Library or Excel 15 Object Library, etc.[1].

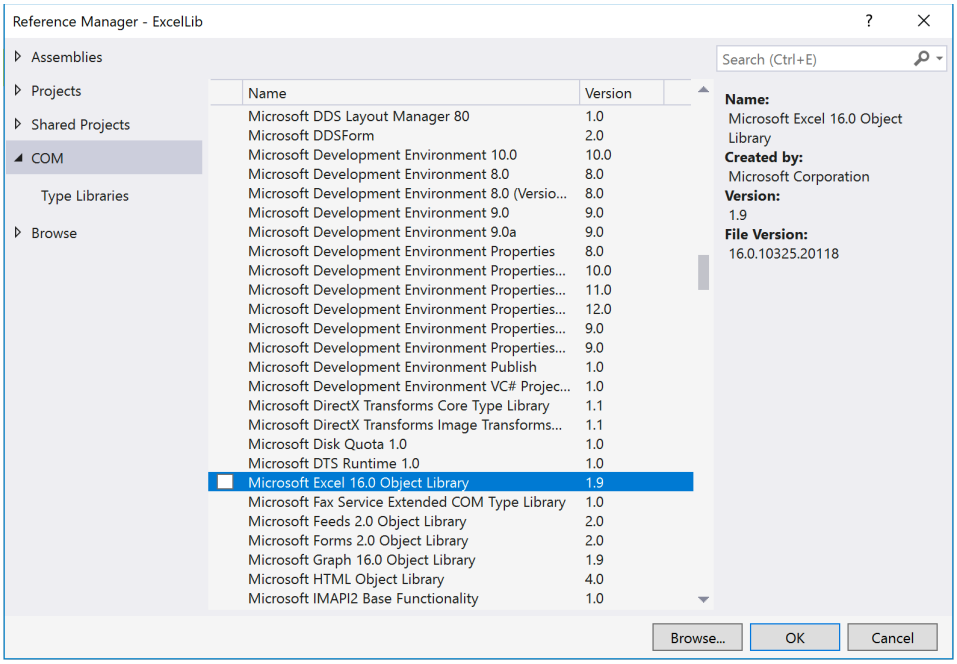

Figure 3 - Add a reference to the Excel Object Library. The version depends on the MS Office version installed

1.2. Add the following directives to the top of the code (**ExcelHandler** class):

```
//create an alias to the namespace/type
using Excel = Microsoft.Office.Interop.Excel;
```

Classes used:

- Excel.Application - Top level object in the Excel object model, used to specify application level properties and application level methods;
- Excel.Worbook - Represents a single workbook within the Excel application;
- Excel.Worksheet - A member of the Worksheets collection in the Workbook object.

[1] The version of the Excel object library may change depending on the excel version installed in the computer.

1.3. Add a static method to create new empty excel files.

```
public static void CreateNewExcelFile(string filename)
{
    //Creates and Excel Application instance
    var excelAplication = new Excel.Application();
    excelAplication.Visible = true;

    //Creates an Excel Workbook with a defaut number of sheets.
    var excelWorkbook = excelAplication.Workbooks.Add();
    excelWorkbook.SaveAs(filename, AccessMode: Excel.XlSaveAsAccessMode.xlNoChange);

    //"eliminates" the instances
    excelWorkbook.Close();
    excelAplication.Quit();

    //It's necessary to free all the memory used by the excel objects. e.g.:
    //System.Runtime.InteropServices.Marshal.ReleaseComObject(excelWorkbook)
    //excelWorkbook = null;
    //...
    //GC.Collect();
}
```

1.3.1. Add a static method name *ReleaseCOMObjects* to kill all COM classes used.

1.4. Add a static method to open and write data into an existing excel file. You may write in the file created previously.

```
public static void WriteToExcelFile(string filename)
{
    Excel.Application excelAplication = new Excel.Application();
    excelAplication.Visible = true;

    //Opens the excel file
    Excel.Workbook excelWorkbook = excelAplication.Workbooks.Open(filename);
    Excel.Worksheet excelWorksheet = excelWorkbook.ActiveSheet;

    excelWorksheet.Cells[1, 1].Value = "Hello";
    excelWorksheet.Cells[1, 2].Value = "World!";

    //you may also use the get_Item(1) method where '1' is the worksheet number
    Excel.Worksheet excelWorksheet2 = excelWorkbook.Worksheets.Add();
    excelWorksheet2.Cells[1, 1].Value = "Goodbye";
    excelWorksheet2.Cells[1, 2].Value = "world!";

    excelWorkbook.Save();
    excelWorkbook.Close();
    excelAplication.Quit();

    //Don't forget to free the memory used by the excel objects
    //....
}
```

Note: Instead of the Add() method from the Worksheet class you may use the get_Item(index) method.

2. Add a static method to open and read data from an existing excel file.

```
public static string ReadFromExcelFile(string filename)
{
    var excelAplication = new Excel.Application();
    excelAplication.Visible = false;

    //Opens the excel file
    var excelWorkbook = excelAplication.Workbooks.Open(filename);
    var excelWorksheet = (Excel.Worksheet) excelWorkbook.ActiveSheet;

    string content = excelWorksheet.Cells[1, 1].Value;
    content += (excelWorksheet.Cells[1, 2] as Excel.Range).Text;

    excelWorkbook.Close();
    excelAplication.Quit();

    //Don't forget to free the memory used by the excel objects
    //....

    //release memory from COM Objects
    return content;
}
```

2.1. Add a new **Windows Form App (.Net Framework)** to your solution and name it 'WinExcelApp'. Add a reference to 'ExcelLib' and create a user interface (*Form*) to invoke the methods defined in the previous exercises.

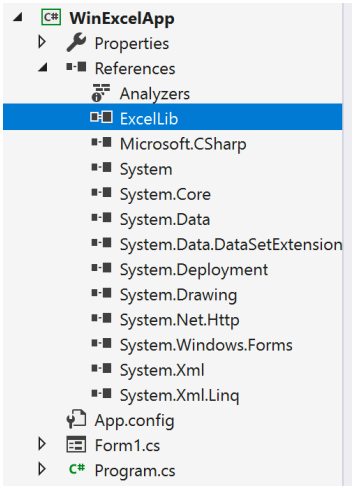

Figure 4 – Windows Form solution structure

3. Add a static method in the library to create a chart.

```
//Add a chart object
Excel.Chart myChart = null;
Excel.ChartObjects charts = excelWorksheet.ChartObjects();
Excel.ChartObject chartObj = charts.Add(50, 50, 300, 300); //Left; Top; Width; Height
myChart = chartObj.Chart;

//set chart range -- cell values to be used in the graph
Excel.Range myRange = excelWorksheet.get_Range("B1:B4");
myChart.SetSourceData(myRange);

//chart properties using the named properties and default parameters functionality in
//the .NET Framwwork
myChart.ChartType = Excel.XlChartType.xlLine;
myChart.ChartWizard(Source: myRange,
    Title: "Graph Title",
    CategoryTitle: "Title of X axis... ",
    ValueTitle: "Title of y axis...");

excelWorkbook.SaveAs(filename);
excelWorkbook.Close();
excelApplication.Quit();
```

4. Call the chart method from the windows form application.

5. Improve your '**ExcelLib**' project with methods to manipulate excel files data. Try those methods using the 'Forms' project.

   5.1. A method to read NxM cells of the first worksheet (N and M must be arguments of the method).

   5.2. A method to read data from a given worksheet (the worksheet identification must be an argument of the method).

   5.3. A method to write a set of data received as an argument.

   5.4. A method that calculates a worksheet's number of lines (with data).

   5.5. A method that searches a specific string in a worksheet (the string must be an argument of the method).

   5.6. Add exception handling to your methods. Make sure the objects are released if an exception occurs.

More information:

Information about how to Access Office Interop Objects by Using Visual C# Features (C# Programming Guide). This may change depending on the C# version: https://msdn.microsoft.com/en-us/library/vstudio/dd264733(v=vs.100).aspx

More information about Microsoft.Office.Interop.Excel namespace at http://msdn.microsoft.com/en-us/library/office/microsoft.office.interop.excel(v=office.15).aspx