



COMPSCI 280 S1 2011 Enterprise Software Development

Developing Windows Applications with Visual Studio 2010
Excel Automation



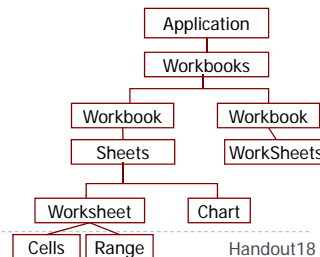
Agenda & Reading

- ▶ **Agenda:**
 - ▶ Excel Automation
 - ▶ Creating Excel Application, Excel Workbook, Excel Worksheet
 - ▶ Manipulating Excel Range
 - ▶ AutoFill and Sorting
 - ▶ Manipulating Excel Cells
- ▶ **Recommended Reading:**
 - ▶ Excel Object Model Overview
 - ▶ [http://msdn2.microsoft.com/en-us/library/wss56bz7\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/wss56bz7(VS.80).aspx)
 - ▶ Excel Tasks
 - ▶ [http://msdn2.microsoft.com/en-us/library/syvd7czh\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/syvd7czh(VS.80).aspx)
 - ▶ Microsoft.Office.Tools.Excel Namespace
 - ▶ [http://msdn2.microsoft.com/en-us/library/microsoft.office.tools.excel\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/microsoft.office.tools.excel(VS.80).aspx)
- ▶ **Namespace covered:**
 - ▶ Microsoft.Office.Interop.Excel
- ▶ **Hands-on Lab**
 - ▶ Lecture18Lab



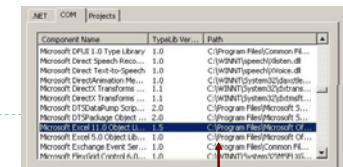
Overview

- ▶ The Application object provides a wrapper around the entire application, and each Workbook object contains a collection of Worksheet objects. From there, the major abstraction representing cells is the Range object, which allows you to work with individual cells or groups of cells.
 - ▶ The application object represents the Excel application itself.
 - ▶ The Workbooks collection makes it possible to work with all the open workbooks, create a new workbook, and import data into a new workbook.
 - ▶ The workbook class represents a single workbook within the Excel application. It also provides a huge number of properties.
 - ▶ The Worksheet Class allows you to work with an individual worksheet
 - ▶ A Range object represents a cell, a row, a column, a selection of cells containing one or more blocks of cells (which may or may not be contiguous), or even a group of cells on multiple sheets.
- ▶ To automate an Excel application object
 - ▶ Create an application object
 - ▶ Manipulate by references to that object's child objects
 - ▶ Create a new workbook
 - ▶ Create a new worksheet/ refer to an existing worksheet
 - ▶ Manipulate worksheet by using range/cell
 - ▶ Save the file
 - ▶ Quit the application
 - ▶ Release resources



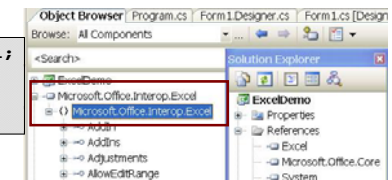
To start ...

- ▶ **Adding Reference**
 - ▶ In order to use a Excel component in your application, you must first add a reference to it.
 - ▶ Create a reference to a type library
 - ▶ Select **Add Reference**
 - ▶ Select **COM** and **Microsoft Excel 11.0 Object Library**.
- ▶ **Note:** You need to add the using directive
 - ▶ Use Namespace alias to provide a shortcut for referring to the Microsoft.Office.Interop.Excel namespace.



```

using Excel = Microsoft.Office.Interop.Excel;
...
Excel.Application xlApp;
xlApp = new Excel.Application();
  
```





Application & Workbooks

Application

- Properties:
 - The **Visible** property determines whether the object is visible.
- To create an object that can reference the objects in an Excel type library

```
xlApp = new Excel.Application();
xlApp.Visible = true;
xlApp.DisplayAlerts = false; //bypass warnings
```

- To quit the application

```
xlApp.Quit();
xlApp = null //release resources
```

Workbooks Collection

- Each Excel application instance has a collection of Workbooks
- Properties
 - The **Count** property returns the number of objects in the collection.
- To create a new workbook

Parameter: Template
Means "use the default value"

```
xlBook = xlApp.Workbooks.Add(Type.Missing); //Create a new workbook
```

- To close all open workbooks
- To open an existing workbook

```
xlApp.Workbooks.Close();
```

```
xlApp.Workbooks.Open(@"Book2.xls", Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing); Handout18
```



Workbook

- The **Workbook** object is a member of the **Workbooks** collection. The **Workbooks** collection contains all the **Workbook** objects currently open in Microsoft Excel.
- Each **Workbook** has a collection of **Worksheets/Charts**
- Properties:
 - The **Name** property returns the workbook name. You cannot set the name by using this property; if you need to change the name, use the **SaveAs** method to save the workbook under a different name.
 - The **Sheets** property returns a **Sheets** collection that represents all the sheets in the specified workbook.
 - The **Charts** property returns a **Sheets** collection that represents all the chart sheets in the specified workbook. Each chart sheet is represented by a **Chart** object.
 - The **Worksheets** property returns a **Sheets** collection that represents all the worksheets in the specified workbook.

```
txtLog.AppendText(xlBook.Name);
```

Methods

- The **Close** method closes the object.
 - Parameter:
 - True** & filename: Saves the changes to the workbook
 - False**: Does not save the changes to this file
 - Omitted**: Displays a dialog box asking the user whether or not to save changes.
- The **Save** method saves changes to the specified workbook.
- The **SaveAs** method saves changes to the workbook in a different file.
 - The first time you save a workbook, use the **SaveAs** method to specify a name for the file.

```
xlBook.Close(true, "ExcelDemo.xls", Type.Missing);
```



Sheets, Worksheets & Charts

Sheets Collection

- The **Sheets** collection can contain **Chart** or **Worksheet** objects.
- The **Sheets** collection is useful when you want to return sheets of any type.

- Properties:
 - The **Count** property returns number of objects in the collection
 - The **Item** property returns a single object from a collection.

```
foreach (Excel.Worksheet xls in xlBook.Sheets) {
    txtLog.AppendText(xls.Name + " ");
}
```

Worksheets Collection

- A collection of all the **Worksheet** objects in the specified or active workbook.
- Each **Worksheet** object represents a worksheet.

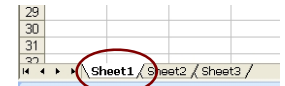
```
foreach (Excel.Worksheet xls in xlBook.Worksheets) {
    txtLog.AppendText(xls.Name + " ");
}
```

Charts Collection

- A collection of all the chart sheets in the specified or active workbook.
- Each chart sheet is represented by a **Chart** object. This doesn't include charts embedded on worksheets or dialog sheets.



Worksheet



Each Worksheet object represents a worksheet.

- Properties
 - The **Name** property gets or sets the name of the worksheet.
 - The worksheet name is shown on the tab for the worksheet.
 - The **Range** property gets a **Excel.Range** object that represents a cell or a range of cells.
 - The **Cells** property gets a **Range** object that represents all the cells on the worksheet (not just the cells that are currently in use).

Methods

- To return a single **Worksheet** object in a workbook
 - Use **Worksheets[index]**, where **index** is the worksheet index number or name, or
 - Use **Sheets[index]**
 - It returns an **Object**, as opposed to a **Worksheet**, as you might expect, we have to use casting to do a conversion

```
xlSheet1 = (Excel.Worksheet)xlBook.Sheets[1];
xlSheet2 = (Excel.Worksheet)xlBook.Worksheets[1];
```

- The **select** method selects the worksheet

```
xlSheet1.Select(Type.Missing);
```



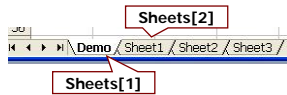
Worksheet (con't)

Method

- The Add method creates a new sheet and adds it to the collection. The new worksheet becomes the active sheet.
- To add a new worksheet
 - Use Worksheets.Add(), or
 - Use Sheets.Add()

```
xlSheet1 = (Excel.Worksheet)xlBook.Worksheets.Add(Type.Missing, Type.Missing, Type.Missing, Type.Missing);
xlSheet1.Name = "Demo";
```

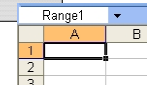
- Note: The worksheet index number denotes the position of the worksheet on the workbook's tab bar.
 - Worksheets[1] is the first (leftmost) worksheet in the workbook, and
 - Worksheets[Worksheets.Count] is the last one.
 - All worksheets are included in the index count, even if they're hidden



Range

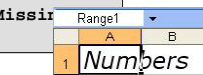
- Represents a cell, a row, a column, a selection of cells containing one or more contiguous blocks of cells, or a 3-D range.
 - To return a Range object that represents a single cell or a range of cells in a worksheet
 - Use get_Range(arg, Type.Missing), where arg names the range
 - If you use a text argument for the range address, you must specify the address in A1-style notation (you cannot use R1C1-style notation).

```
Excel.Range xlRange = xlSheet1.get_Range("A1", Type.Missing);
xlRange.Name = "Range1";
xlRange.set_Value(Type.Missing, "Numbers");
```



- Properties/Methods:
 - The Name property returns or sets the name of the object.
 - The Set_value method sets the value of the specified range.
 - The Formula property returns or sets the object's formula in A1-style notation and in the language of the macro.
 - The Font property returns a Font object that represents the font of the specified object.
 - To refer to a named range

```
Excel.Range xlNamedRange = xlSheet1.get_Range("Range1", Type.Missing);
xlNamedRange.Font.Name = "Verdana";
```



Excel uses the A1 reference style, which refers to columns with letters and refers to rows with numbers. These letters and numbers are called row and column headings. To refer to a cell, enter the column letter followed by the row number. For example, B2 refers to the cell at the intersection of column B and row 2. A10:E20 refers to the range of cells in columns A through E and rows 10 through 20.



Range (con't)

Array:
3 rows x 5 columns

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6

To fill a range with an Array

```
int[,] iaNumbers = new int[3, 5];
for (iRow = 0; iRow < 3; iRow++){
  for (iCol = 0; iCol < 5; iCol++){
    iaNumbers[iRow, iCol] = iRow + iCol;
  }
}
```

	A	B	C	D
1	Numbers			
2	0	1	2	3
3	1	2	3	4

```
Excel.Range xlRectRange = xlSheet1.get_Range("A2", "D3");
xlRectRange.set_Value(Type.Missing, iaNumbers);
```

5	10	10
6	10	10

To specify the "corners" of a range

```
xlRectRange = xlSheet1.get_Range("A5", "B6");
xlRectRange.set_Value(Type.Missing, 10);
```

To refer to a particular row or column or range of rows and columns

```
xlRangeByRow = (Excel.Range)xlSheet1.Rows[3, Type.Missing];
xlRangeByRow.Font.Italic = true;
xlRangeByCol = (Excel.Range)xlSheet1.Columns[4, Type.Missing];
xlRangeByCol.Font.Bold = true;
```

0	1	2	3
1	2	3	4

Italic

Bold

- To fill a multi-cell range without populating cells one at a time, you can set the Value property of a Range object to a two-dimensional array.
- Note: iaNumbers is a three-by-five array. But the rectangular region is a two-by-four region.



Range (con't)

Original Range = E1

	E	F	G	H
1				
2				
3				

Offset(1,1)

Offset(2,2)

Offset(3,3)

- To fill a range using the Offset property
 - Use the Offset property of a range to retrieve a range relative to the original range

```
Excel.Range xlRangeOffset = xlSheet1.get_Range("E1", Type.Missing);
for (int i = 1; i < 6; i++) {
  xlRangeOffset.get_Offset(i, i).set_Value(Type.Missing, i);
}
```

To fill a range using the Formula property

- Formulas are equations that perform calculations on values in your worksheet. A formula starts with an equal sign (=).
- "\$A\$1" (the cell at location A1) and "\$A\$1:\$C\$5" (the range consisting of all cells within the rectangle bounded by A1 and C5). The "\$" indicates an absolute, as opposed to relative, coordinate.
- R2C4:R3C4 refers to the cells in the rectangle region with one corner at the 2nd row 4th column and another corner at the 3rd row and 4th column (\$D\$2:\$D\$3)

```
xlSheet1.get_Range("A4", Type.Missing).Formula = "=Count(A2:A3)";
xlSheet1.get_Range("B4", Type.Missing).Formula = "=SUM(B2:B3)";
xlSheet1.get_Range("C4", Type.Missing).Formula = "=SUM(C2:C3)/$A$5";
xlSheet1.get_Range("D4", Type.Missing).Formula = "=AVERAGE(R2C4:R3C4)";
```

=AVERAGE(\$D\$2:\$D\$3)	
C	D
2	3
3	4
0.5	3.5

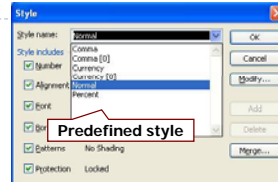


Style

- To apply a predefined style to a workbook range

```
xlRange.Style = xlBook.Styles["Currency"];
```

- Check the predefined styles in Excel
 - Select Format | Style ...
 - Comma, Currency, Percent, etc



- To create a new style to a workbook range

```
Excel.Style sty = xlBook.Styles.Add("Style1", Type.Missing);
sty.Font.Name = "Verdana";
sty.Font.Bold = true;
xlRangeNewStyle.Style = sty;
```

\$	1.00	1
\$	2.00	3
\$	3.00	5
\$	4.00	7
\$	5.00	9

- To clear the styles in a workbook range

- Use the Style property of a range object
 - The style property returns a Style object that represents the style of the specified range.

```
xlRange.Style = xlBook.Styles["Normal"];
```



Range (con't)

7	
8	Monday
9	Tuesday
10	Wednesday
11	Thursday
12	Friday
13	Saturday
14	Sunday
15	
16	

- Methods

- The **AutoFit** method changes the width of the columns in the range or the height of the rows in the range to achieve the best fit.
 - The expression must be a row or a range of rows, or a column or a range of columns. Otherwise, this method generates an error.

```
xlRangeByCol.AutoFit();
```

- The **ClearContents** method clears the formulas from the range.
- The **ClearFormats** method clears the formatting of the object.
- The **UnMerge** method separates a merged area into individual cells.
- The **Merge** method creates a merged cell from the specified Range object.

```
xlSheet1.get_Range("A1", "B1").Merge(Type.Missing);
```

Range1		Numbers
A	B	C
1	Numbers	

- The **AutoFill** method Performs an autofill on the cells in the specified range.
- The **Sort** method sorts a range



AutoFill

- The AutoFill method is used to store incrementally increasing or decreasing values into a range.

- You can specify the behaviour by supplying an optional constant from the XlAutoFillType enumeration (xlFillDays, xlFillMonths, xlFillSeries etc.)
- You must specify two ranges: The range calling the AutoFill method, which specifies the "starting point" of the fill and the range to be filled, passed as a parameter to the AutoFill method;
 - Note: the destination range must include the source range.

8	Monday	Jan
9	Tuesday	Feb
10	Wednesday	Mar
11	Thursday	Apr
12	Friday	May
13	Saturday	Jun
14	Sunday	Jul
15		Aug
16		Sep
17		Oct
18		Nov
19		Dec
20		

- To fill a range with values automatically

```
Excel.Range sourceRange = xlSheet1.get_Range("A8", Type.Missing);
sourceRange.set_Value(Type.Missing, "Monday");
sourceRange.AutoFill(xlSheet1.get_Range("A8:A14", Type.Missing),
Excel.XlAutoFillType.xlFillDays);
```

```
sourceRange = xlSheet1.get_Range("B8", Type.Missing); //Fill by months
sourceRange.set_Value(Type.Missing, "Jan");
sourceRange.AutoFill(xlSheet1.get_Range("B8:B19", Type.Missing),
Excel.XlAutoFillType.xlFillMonths);
```

```
sourceRange = xlSheet1.get_Range("C8", Type.Missing); //Fill by series
sourceRange.set_Value(Type.Missing, 1);
sourceRange.AutoFill(xlSheet1.get_Range("C8:C12", Type.Missing),
Excel.XlAutoFillType.xlFillSeries);
```



AutoFill (con't)

1	1	2
2	3	5
3	5	8
4	7	11
5	9	14

- To fill a range by Series

- 1, 2, 3 ...
- 1, 3, 5 ...

```
sourceRange = xlSheet1.get_Range("C8", Type.Missing);
sourceRange.set_Value(Type.Missing, 1);
sourceRange.AutoFill(xlSheet1.get_Range("C8:C12", Type.Missing),
Excel.XlAutoFillType.xlFillSeries);
```

```
xlSheet1.get_Range("D8", Type.Missing).set_Value(Type.Missing, 1);
xlSheet1.get_Range("D9", Type.Missing).set_Value(Type.Missing, 3);
sourceRange = xlSheet1.get_Range("D8:D9", Type.Missing);
sourceRange.AutoFill(xlSheet1.get_Range("D8:D12", Type.Missing),
Excel.XlAutoFillType.xlFillSeries);
```

- To fill a range by Formula

```
sourceRange = xlSheet1.get_Range("E8", Type.Missing); //Fill by formulas
sourceRange.Formula = "=SUM(C8:D8)";
sourceRange.AutoFill(xlSheet1.get_Range("E8:E12", Type.Missing),
Excel.XlAutoFillType.xlFillDefault);
```

- You can quickly fill cells in a range with a series of numbers. For example, if you want the series 1, 2, 3, 4, 5..., type 1 in the first cell. If you want the series 2, 4, 6, 8..., type 2 and 4 in the first two cells. If you want the series 2, 2, 2, 2..., type 2 in the first two cells.
- If this argument is xlFillDefault or omitted, Microsoft Excel selects the most appropriate fill type, based on the source range.



Cells

- ▶ Use Cells[row, column], where row is the row index and column is the column index, to return a single cell.

- ▶ To fill a range using the Cells property and a For-Next Loop

```
for (iCol = 3; iCol <= 5; iCol++){  
  for (iRow = 14; iRow <= 16; iRow++){  
    xlSheet1.Cells[iRow, iCol] = Generator.Next(10);  
  }  
}
```

10	2	9
0	4	5
9	9	0

- ▶ To set a value

```
xlSheet1.Cells[14, 3] = 10;
```

Row Column

◆ Although you can also use Range("A1") to refer cell A1, there may be times when the Cells property is more convenient because you can use a variable for the row or column.