# Lab 13: Coding with Excel and Excel Services

#### Objectives

After completing this lab, you will be able to:

* Use and publish an Excel workbook designed to that advantage of Excel Services
* Create a Trusted File Location in Excel Services
* Develop and external C# application that will use Excel Services Web Service

#### Prerequisites

Before working on this lab, you must have:

* Labs 1 and 2 fully completed

#### Scenario

This document shows how to build custom .Net code that incorporates server side Excel based logic into an application and calls the Excel Web Services to run the Excel calculation portion of the logic on the server.

Specifically, the sample revolves around a “**Monte Carlo**” simulation model for a stock and option portfolio. The user runs a command line tool and passes some arguments to the command. The code passes the arguments to the server thru Web service method calls, and gets results back. It then outputs the results to the console.

A console application was chosen for this sample because it enables us to easily show a complete application, while focusing on the Excel Web Services aspects, rather than on UI considerations.

Key aspects demonstrated here:

* Spreadsheet based logic incorporated into enterprise server applications.
* Proprietary information (generally the formulas and calculation model) protected on the server. End users can use the model thru the application even if they don’t have access to the model itself.
* Business Analysts maintains the Excel formulas and calculation model portion of the application, without intervention of IT Developer or an Administrator.

##### Estimated time to complete this lab: 60 minutes

### Exercise 1 Publish an Excel workbook to a document library

In this exercise you will use an excel workbook to calculate the total value of X number of stocks and options of a specific company. This workbook was designed to allow an external application to use some of it fields by using labeled cells and the Excel Services Web Service.

1. Create a new Team Site under “**Sites**” and call it “**My Team Site**”
2. Create a new Document Library within that site called “**MonteCarlo**” based on “**MS Office Excel spreadsheet**” Document Template.
3. In the “MonteCarlo” document library click the down arrow beside settings, select “**Document Library Settings**”, under the “**General Settings**” section, select “**Advanced settings**”, locate the “**Browser-enabled Documents**” and select the “**Display as a web page**” bullet. Click **Ok**
4. Return to the document library – click the “MonteCarlo” link in the breadcrumb

### Exercise 2 Enable the new document library as a “Trusted File Location”

Make this location a “**Trusted File Location”** from the point of view of Excel Services

1. In the **Central Administration**, select **Application Management**
2. Select “**Create or configure this farm's shared services**”
3. Click on the SSP – in this case “**O12ServerSSP1 (Default)**” link
4. In the “**Excel Services Settings**” section, click on “**Trusted file locations**”
5. Click “**Add Trusted File Location**” located in the blue bar
6. Paste the URL of the “**MonteCarlo**” document library in the “Address” field. It should look similar to the following: (<http://hol.litwareinc.com/SiteDirectory/MyTeamSite/MonteCarlo>)
7. Keep all other options as is an click **Ok**

**Note:** Only files published to trusted locations can be opened and used by Excel Services.

1. Open the “**MonteCarlo.xlsx”** workbook located at “**C:\hol\MonteCarlo”**.

**Note:** You need Excel 2007 in order to publish the workbook appropriately to the server. Version 2003 or older of Excel does not have that functionality.

1. Notice the “**Simulation**” sheet!

The Simulation sheet is conceptually the proprietary calculation model, which we want to protect on the server and use in an application. For this sample, the model is “fake” – just a few randomly generated numbers – but the workbook interface to the application is demonstrated just as if this was a real elaborated calculation model.

1. In the File menu (Office logo top left hand corner), select “**Publish”,** then “**Excel Services”.**  The “**Save as**” dialog opens.
2. Just above the “**Save**” button, click the “**Excel Services Options**” button

**Note:** A good practice would be to uncheck the **Simulation** sheet to protect your calculation model and make it unavailable for viewing on the server. The entire file is still copied over to the server location and available to the calculation components of the server, but only users with extended rights can view this sheet or open the entire file in Excel.

1. Click **Ok**
2. **Uncheck** the “**Open this workbook in my browser after I save**” box
3. **Browse** to the “**MonteCarlo”** document library by selecting the “My Network Places” in the left hand side panel i.e. “**http://hol.litwareinc.com/SiteDirectory/MyTeamSite/MonteCarlo/**” and click “**save”** to save your workbook at that location with the default file name.
4. Your file is now published to a server document library. If you browse to the library, you’ll see your file listed.
5. Your Browser will launch and will show the MonteCarlo.xlsx file. Notice that the Simulation tab is no available.
6. Close Excel and the browser session.

### Exercise 3 Developing the Monte Carlo Application

This C# external application uses Excel Services Web Service and its calculation capabilities (formulas in the MonteCarlo spreadsheet) by passing four values – the company Symbol, the initial price, the strike price and lastly the interest rate – and wait for the calculated results which are display onto the console. You will build this application!

* Create a C# project

1. Launch Visual Studio 2005 from the Desktop link
2. Create a new project of type **Visual C# /** **Windows** and template “**Console Application” c**alled “**MonteCarlo**”
3. Save the project at the following location: C:\HOL\MonteCarlo (browse to that location and click **Ok**.
4. Add a web reference to the Excel Web Services:

[**http://hol.litwareinc.com/\_vti\_bin/excelservice.asmx**](http://hol.litwareinc.com/_vti_bin/excelservice.asmx)

**Note1:** Visual Studio creates a proxy class for accessing the Web service. The class is called ExcelService, and it is in the namespace <your-app><server-name>. In our example, the namespace is: **MonteCarlo.com.litwareinc.hol**

**Note2:** The Excel Services Server name is “**hol.litwareinc.com**”

1. Add a using statement to the new namespace:  
    using MonteCarlo.com.litwareinc.hol;
2. Define the interface between your Console Application and the Excel workbook. The interface is a set of named ranges that are associated with input and output cells. Insert the following code:

class Program

{

// input & output cell names

private static string[] inputCells = {"Symbol", "InitialPrice", "StrikePrice", "InterestRate"};

private static string[] outputCells = {"Average", "VaR", "RelDeltaNormal"};

**Note:** Both direct range references (e.g. “B52”) and named ranges can be used. Using named ranges isolates the parameter location in the workbook from the application, thus making the model more robust and easier to maintain.

1. Add code to **display** a usage line when the command is run with an insufficient number of arguments.

static void Main(string[] args)

{

if (args.Length < 1 + inputCells.Length)

{

Console.Error.Write("Usage: MonteCarlo <WorkbookPath>");

foreach (string argName in inputCells)

Console.Error.Write(" <{0}>", argName);

Console.Error.WriteLine("");

return;

}

**Note:** The usage line will help you in run-time. It mentions all required arguments – a path to the server-side Excel workbook and the input cell names that you defined in the previous step.

1. Add code to instantiate and initialize the proxy object.

// Excel web services stuff:

// Instantiate and initialize the service proxy object

ExcelService xlSrv = new ExcelService();

xlSrv.Credentials = System.Net.CredentialCache.DefaultCredentials;

**Note:** Setting credentials to **DefaultCredentials** means that your process security context is passed along to the web service for authentication.

1. Open the workbook and start a session with Excel Services.

// Open the workbook

Status[] status;

string sessionId = xlSrv.OpenWorkbook(args[0], string.Empty, string.Empty, out status);

**Note:** The **OpenWorkbook** method returns a session ID, which is used in subsequent calls to the server as your session context.

The two empty strings designate default cultures (languages) to be used – the context of your process.

The “**status**” output argument returns an array of non-critical errors (usually empty). Critical errors are surfaced as SOAP exceptions that you can catch.

1. Set the values that were passed on the command line into cells in the workbook, using the array of input named ranges that you defined above.

// Set parameter values into cells

for (int i = 0; i < inputCells.Length; i++)

status = xlSrv.SetCellA1(sessionId, "Sheet1", inputCells[i], args[1+i]);

1. Call a method to make the server calculate the new state of the workbook.

// Calculate the workbook.

status = xlSrv.CalculateWorkbook(sessionId, CalculateType.Recalculate);

**Note:** When the model is complex and time consuming to calculate, it is often a good idea to turn automatic recalculation off while authoring the workbook in Excel. When automatic recalculation is turned off, the server will not recalculate after each parameter value that you set, and you need to explicitly instruct the server to recalculate once all parameters are set.

1. Once the workbook is calculated, you can get output values from the server session, using the array of output named ranges that you defined above, and write them out.

// Get results and write them out.

foreach (string cellName in outputCells)

{

object result = xlSrv.GetCellA1(sessionId, "Sheet1", cellName, true, out status);

Console.WriteLine("{0}:\t{1}", cellName, result);

}

1. Lastly, close the workbook and your server session.

// Close the session.

status = xlSrv.CloseWorkbook(sessionId);

**Note:** *This is a very important optimization step. If you do not close the session when you are done with it, the server will still time it out after an administrator-controlled amount of time; but this timeout is usually set according to the needs of GUI-based applications, to allow users to pause between interactions. Unnecessary sessions that remain open consume server resources and may cause decreased overall performance*.

1. Your application code is ready! Now have Visual Studio build it, and the development phase is done. On the menu, select **Build**, then **Build MonteCarlo.**

The results of the build should be:

========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========

### Exercise 4 Running the Application

In this exercise you will test your external C# application along with the Excel Services Web Service.

1. Open a command window: Start, command Prompt
2. Change to the Debug directory of your newly built project: “C:\HOL\MonteCarlo\MonteCarlo\bin\Debug”
3. Run your program without arguments to get the usage line. On the command line, enter: “MonteCarlo”

You will be returned information about the usage of this program

1. Run your program, giving it the path to the server Excel workbook, and values to the other arguments. Notice the three result values that are written out. Enter: “**MonteCarlo “http://hol.litwareinc.com/SiteDirectory/MyTeamSite/MonteCarlo/MonteCarlo.xlsx” MSFT 26 31 5%**” at the command line.
2. Try with different numbers
3. *Do not close the console yet…* From the server document library, open the workbook for editing in Office Excel 2007. Click on the file’s context menu then, select **Edit in Microsoft Office Excel**.
4. Once in Excel, select cell **A15** of Sheet1, where the **Average** output value is. Notice its value.
5. Modify the cell and multiply it by 10. The resulting calculation should look like =Simulation!B3\*10.
6. Then save the file and quit Excel.
7. In the command window again, simply run the last command once more, to execute your program again with the same set of input values. Notice how the **Average** result is now an order of magnitude larger!
8. You made this change in the algorithm without touching a line of code – just modifying the Excel calculation model.

Close all applications - this lab is completed!