

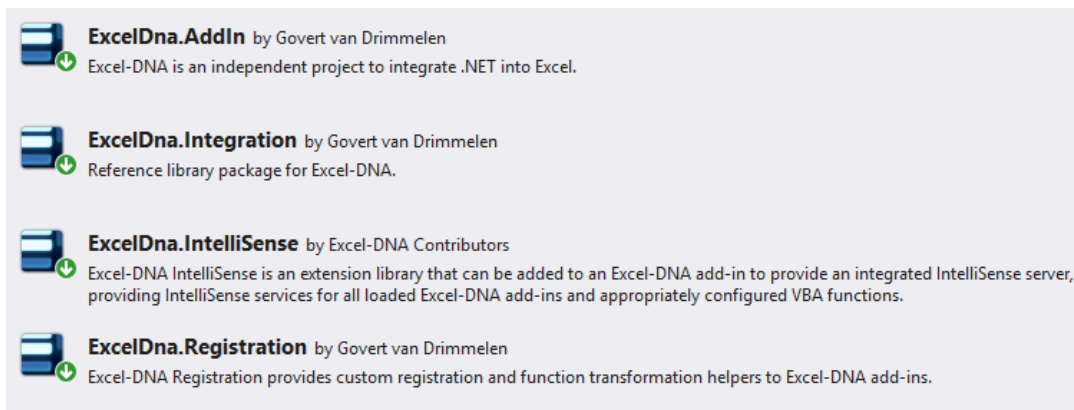
## Dynamically create and load user defined functions (UDF) for excel using Excel-DNA

Recently I faced a challenge of developing an excel plugin that can interact with a Java calculation server. The Java server was able to dynamically load JARs that contained all the calculations and make them available for invocation through a REST endpoints. If you wish to develop a Java server like this then [this post](#) might be helpful. It also had an endpoint to get the list of all the calculations and required parameters. The excel plugin was supposed to call the server and register all the calculations with excel so that they could easily be called from any cell.

I decided to develop the plugin using Excel-DNA. I have written this detailed article on [how to develop excel plugins](#) using this library. Therefore, the only additions required were to somehow dynamically register new calculations in excel and adding a HttpClient to make REST calls.

### Setting up the project

Apart from the minimum Excel-DNA setup as described in this article, you need to add the following Nuget packages:



The screenshot displays four NuGet packages available for installation, each with an Excel icon and a green download arrow:

- ExcelDna.AddIn** by Govert van Drimmelen  
Excel-DNA is an independent project to integrate .NET into Excel.
- ExcelDna.Integration** by Govert van Drimmelen  
Reference library package for Excel-DNA.
- ExcelDna.IntelliSense** by Excel-DNA Contributors  
Excel-DNA IntelliSense is an extension library that can be added to an Excel-DNA add-in to provide an integrated IntelliSense server, providing IntelliSense services for all loaded Excel-DNA add-ins and appropriately configured VBA functions.
- ExcelDna.Registration** by Govert van Drimmelen  
Excel-DNA Registration provides custom registration and function transformation helpers to Excel-DNA add-ins.

And then the initial code in the IExcelAddin implementation would be like this

```
1 public void AutoOpen()  
2 {  
3     try
```

```

9         xlApp.StatusBar = "Functions registered";
10    }
11    catch (Exception ex)
12    {
13        xlApp.StatusBar = "Error in registering functions";
14    }
15 }
16
17 public void AutoClose()
18 {
19     IntelliSenseServer.Uninstall();
20 }
21
22 public void RegisterFunctions()
23 {
24 }

```

## ExcelDNA.Registration

Next step is to implement our RegisterFunctions block to handle all function registration. Let us suppose our calculation service sends calculation information as JSON as per following contract

```

1 public class CalcInfo
2 {
3     public string name { get; set; }
4     public string description { get; set; }
5     public ArgInfo[] inputParams { get; set; }
6 }
7
8 public class ArgInfo
9 {
10    public string name { get; set; }
11    public string type { get; set; }
12    public string description { get; set; }
13 }

```

Then the entire system for implementing the registration would be as coded below:

```

1 public void RegisterFunctions()
2 {
3     try
4     {
5         // Get all calculation details
6         List<CalcInfo> calculations = Service.Instance.GetAllCalculations();
7
8         // Create function registration entries using ExcelDNA.Registration
9         IEnumerable<ExcelFunctionRegistration> calcEntries = calculations.Select(c
10    {
11        // Create ExcelFunctionAttribute for function hints in excel
12        ExcelFunctionAttribute funcAttr = new ExcelFunctionAttribute()
13        {
14            Name = calc.name,
15            Description = calc.description
16        };
17
18        // Create parameter registration entries for parameter hint
19        List<ExcelParameterRegistration> paramEntries = calc.inputParams.Select
20            new ExcelParameterRegistration(new ExcelArgumentAttribute() { Name
21
22        // Create a lambda expression
23        LambdaExpression exp = FuncToExpression(calc);

```

```

29
30     ExcelRegistration.RegisterFunctions(calcEntries);
31 }
32 catch (Exception ex)
33 {
34
35 }
36 }
37
38 /// <summary>
39 /// Converting our calculation into a LambdaExpression to be used by ExcelDNA.Regis
40 /// </summary>
41 /// <param name="calc">The calculation information</param>
42 /// <returns></returns>
43 private LambdaExpression FuncToExpression(CalcInfo calc)
44 {
45     // Add as many cases as the maximum no. of arguments to be supported
46     switch (calc.inputParams.Length)
47     {
48         case 1: return FuncExpression1((a1) => ExecuteCalculation(calc, a1));
49         case 2: return FuncExpression2((a1, a2) => ExecuteCalculation(calc, a1, a2);
50         case 3: return FuncExpression3((a1, a2, a3) => ExecuteCalculation(calc, a1,
51         case 4: return FuncExpression4((a1, a2, a3, a4) => ExecuteCalculation(calc,
52         case 5: return FuncExpression5((a1, a2, a3, a4, a5) => ExecuteCalculation(c
53         default:
54             return null;
55     }
56 }
57
58 // LambdaExpression generators for different no. of arguments
59 public Expression<Func<object, object>> FuncExpression1(Func<object, object> f) {
60 public Expression<Func<object, object, object>> FuncExpression2(Func<object, object
61 public Expression<Func<object, object, object, object>> FuncExpression3(Func<object
62 public Expression<Func<object, object, object, object, object>> FuncExpression4(Fur
63 public Expression<Func<object, object, object, object, object, object>> FuncExpres
64
65 /// <summary>
66 /// Call the web service endpoint with proper request body
67 /// </summary>
68 /// <param name="calc">The information related to the calculation invoked</param>
69 /// <param name="args">Arguments passed by excel</param>
70 /// <returns></returns>
71 private object ExecuteCalculation(CalcInfo calc, params object[] args)
72 {
73     // Your own implementation of executing any calculation by calling the service
74     object result = null;
75
76     return result;
77 }

```

The actual implementation for *ExecuteCalculation* would depend upon the use case. Also, I imagine there could be a smarter way of converting functions to expressions without repeating it for different no. of arguments. I would appreciate your suggestion.

Categories: [EXCEL](#)

Tags: [.Net](#) [C#](#) [Excel](#) [Excel-DNA](#) [ExcelDNA](#) [function server](#) [plugin](#) [UDF](#) [user defined function](#)

[xlloop alternative](#)

0 Comments

## Leave a Reply

Name \*

Email \*

Website

What's on your mind?

☐ Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search ...



## Recent Posts

[Getting started with CAPE-OPEN implementation in .Net](#)

[Socket Programming – TCP Client and Server in Java](#)

[Socket Programming – TCP Client and Server in .Net C#](#)

## Archives

[February 2020](#)

[December 2019](#)

[September 2019](#)

[May 2019](#)

## Categories

## Related Posts

We can easily develop cool XLA/XLAM addins using Excel's native VBA editor, however, they are not enough in complex scenarios. In such scenarios, we would like to take full advantage of the .Net framework. Now [Read more...](#)