Writing your first plug-in for SMath Studio Desktop in VB.NET

[rev.2 | 2018.01.13 | SS ≥ 0.98.6356]

SMath Studio desktop provides the possibility to write plug-ins to extend program's features. The simplest feature you can think to add in the program is probably a *function*, and this is what we will do step-by-step in this tutorial.

First of all, we have to decide our goal. In this plug-in, we will try to create a combinations function that achieves what is shown below:

$$C(n,k) := \frac{n!}{(k!)\cdot(n-k)!}$$

$C(5,3) = 10$

$C(3,5) =$ ▮ **lastError** = "Factorial is defined for real numbers and zero."

The finished function syntax will be in the form: $\mathrm{combin}(n,k) =$ ▮

This tutorial as well as the complete plug-in code can be found in the public SVN repository of SMath Studio: https://smath.info/svn/public/plugins/Tutorials/VB.NET/CombinFunction/

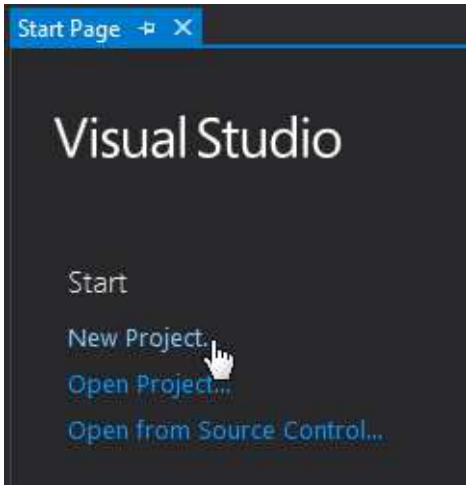
Requirements

To accomplish our task we need an IDE (Integrated Development Environment); you can use the one you want, in this example we will use ***Visual Studio Community 2015*** (you can download it for free on the official website https://www.visualstudio.com/vs/)

The second requirement is to have SMath Studio on your system.

Let's start!

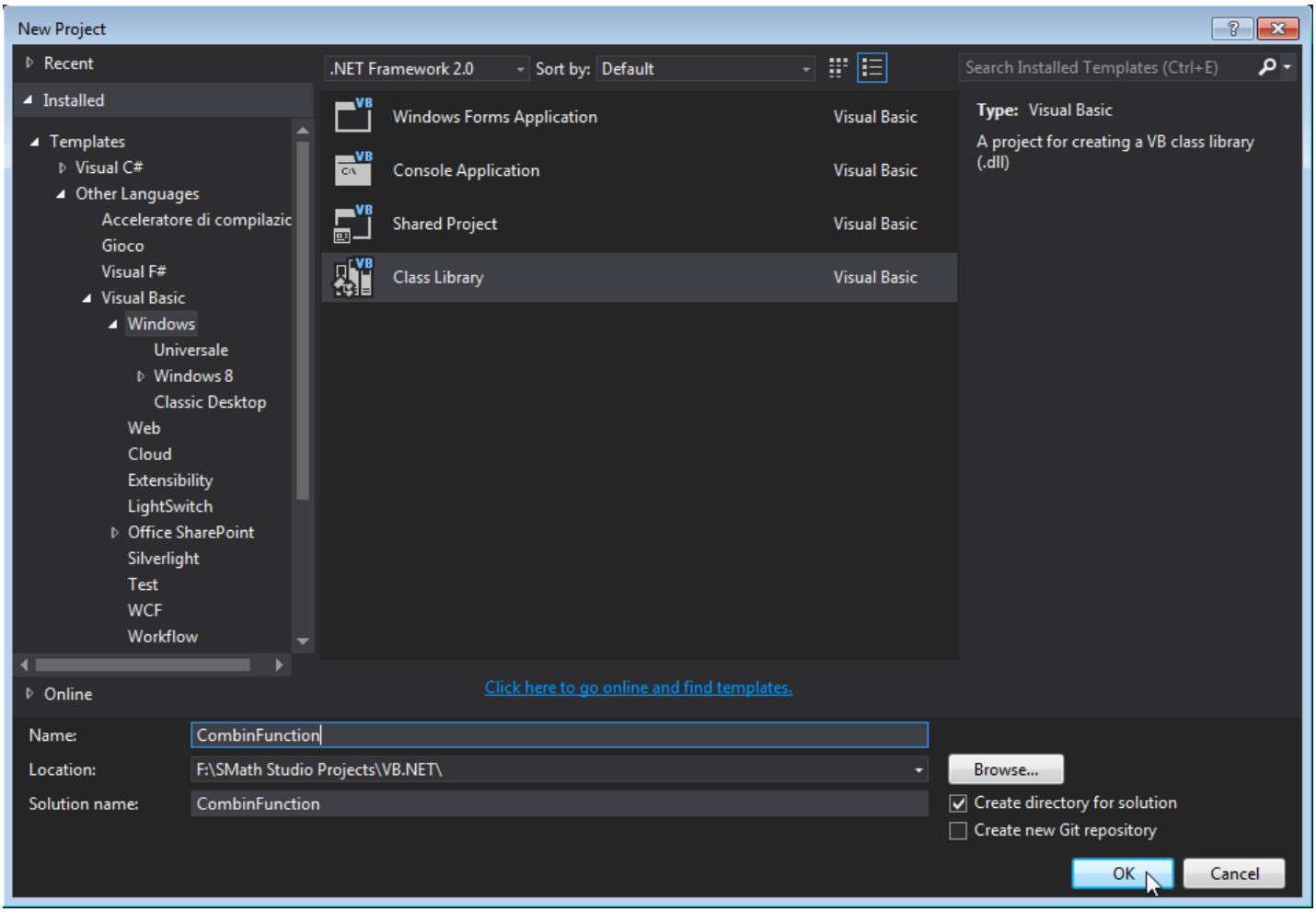
1. Once Visual Studio is installed, open it and click on *File* ⇒ *New Project* from the main menu or *Start* ⇒ *New Project* from the *Start Page*



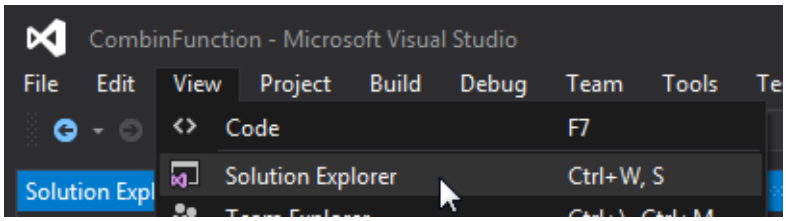
IMPORTANT

Be sure to save your project periodically as you work on this tutorial!

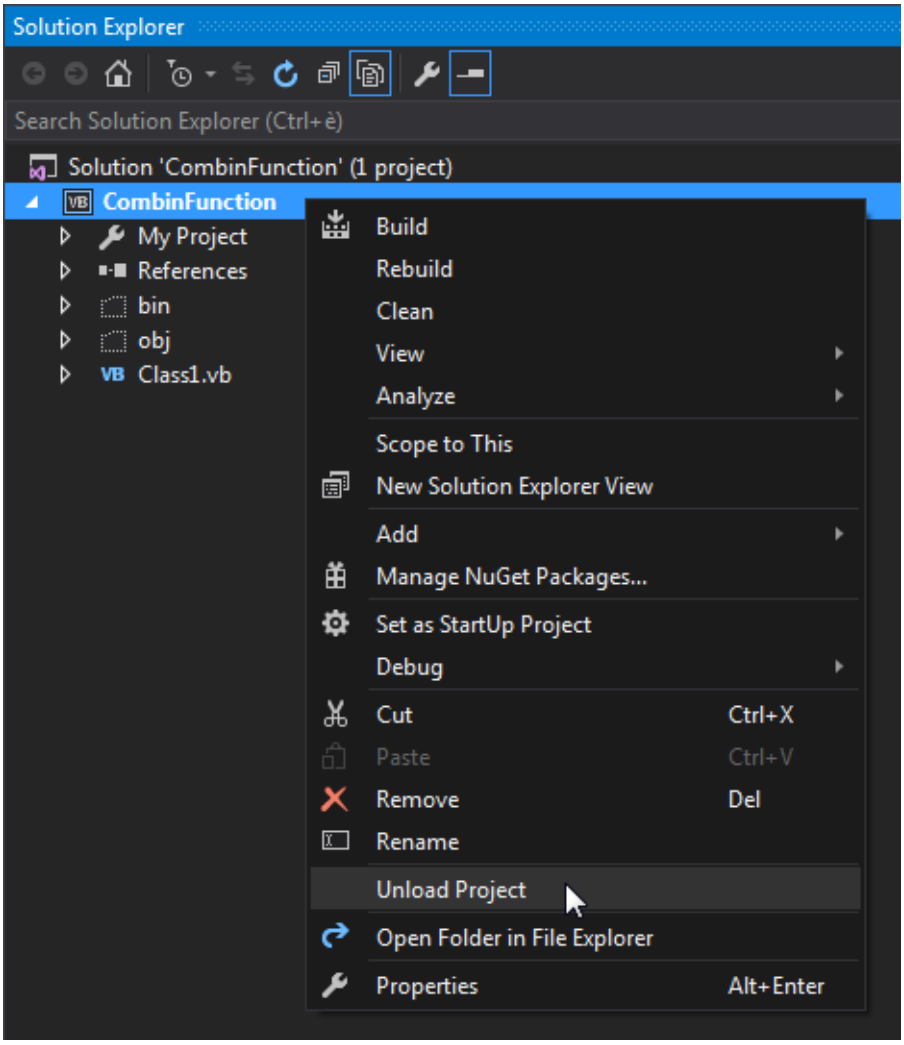
2. In the New Project dialog, choose **.NET Framework 2.0**, then navigate to **Templates** ⇒ **Other Languages** ⇒ **Visual Basic** ⇒ **Windows** ⇒ **Class Library** and type the name for this project.
In this case, we choose **CombinFunction**. Once all is done, click on **OK**.



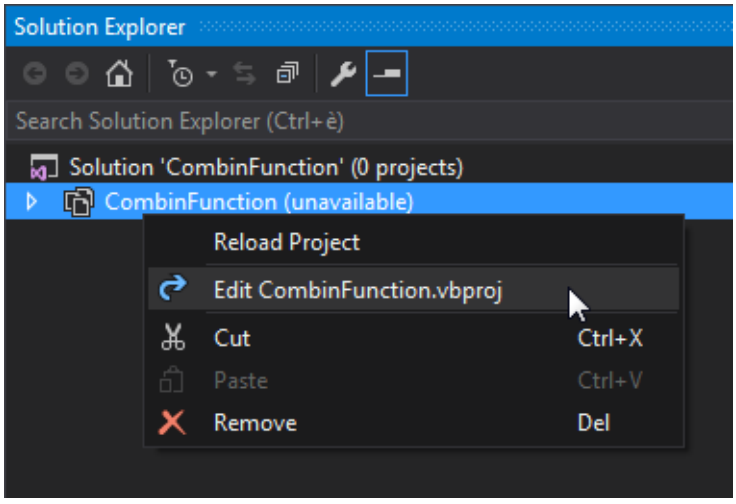
3. Now make the *Solution Explorer* visible (if it is not visible already) by clicking on **View** ⇒ **Solution Explorer**



4. In the *Solution Explorer*, right-click the project name and click **Unload Project**.



5. Now will see *CombinFunction (unavailable)*. Right-click on it and choose **Edit CombinFunction.vbproj**.



6. The project file will be opened. Scroll down to the first *<ItemGroup>* tag and add the following code above it:

```
  <PropertyGroup>
    <!-- Relase -> SMath Release Manager -->
    <SMathDir Condition=" '$(SMathDir)' == '' AND '$(Configuration)' == 'Release'
">..\..\..\Main\SMathStudio\canvas\bin\Debug</SMathDir>
    <!-- Debug -> development -->
    <SMathDir Condition=" '$(SMathDir)' == '' AND '$(Configuration)' == 'Debug'
">C:\Program Files (x86)\SMath Studio</SMathDir>
  </PropertyGroup>
```

hint: you can copy-paste the code from these greyed areas

These lines of code will allow you to have a plug-in ready to be shared with the community, and they let you to compile the plug-in in **Debug** mode on your machine. If is not in your purposes to share the plugin, you can even use the code below instead.

```
  <PropertyGroup>
    <SMathDir Condition=" '$(SMathDir)' == '' ">C:\Program Files (x86)\SMath
Studio</SMathDir>
  </PropertyGroup>
```

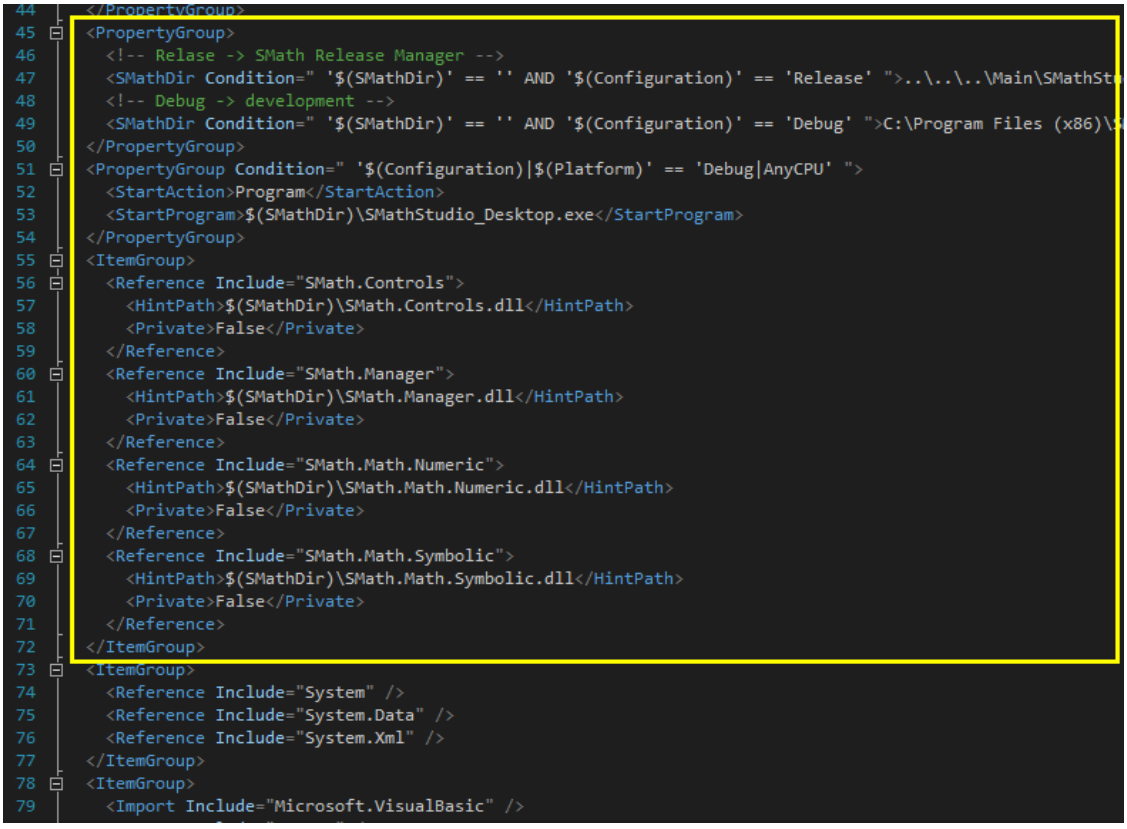
"*C:\Program Files (x86)\SMath Studio*" is obviously the path of SMath Studio on your system (you have to change it if is different).

Under the previous code, add the following code:

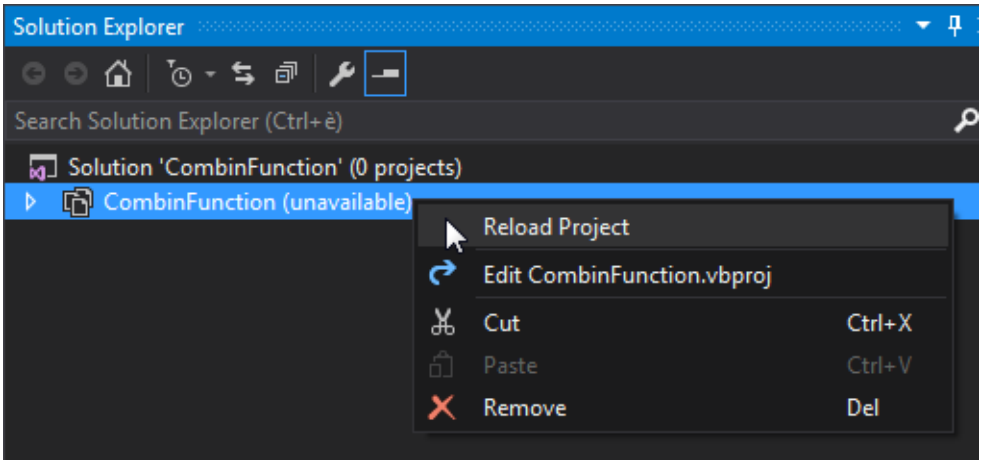
```
  <ItemGroup>
    <Reference Include="SMath.Controls">
      <HintPath>$(SMathDir)\SMath.Controls.dll</HintPath>
      <Private>False</Private>
    </Reference>
    <Reference Include="SMath.Manager">
      <HintPath>$(SMathDir)\SMath.Manager.dll</HintPath>
      <Private>False</Private>
    </Reference>
    <Reference Include="SMath.Math.Numeric">
      <HintPath>$(SMathDir)\SMath.Math.Numeric.dll</HintPath>
      <Private>False</Private>
    </Reference>
    <Reference Include="SMath.Math.Symbolic">
      <HintPath>$(SMathDir)\SMath.Math.Symbolic.dll</HintPath>
      <Private>False</Private>
    </Reference>
  </ItemGroup>
```

This will ensure that the most recent APIs of SMath Studio available on your system will be loaded once you open and compile the project.

Once done, you should see something like in this screenshot. The yellow vertical bar shows the lines of code where there are changes respect to the last save; color becomes olive green after saving to show lines edited since the begin of the session.

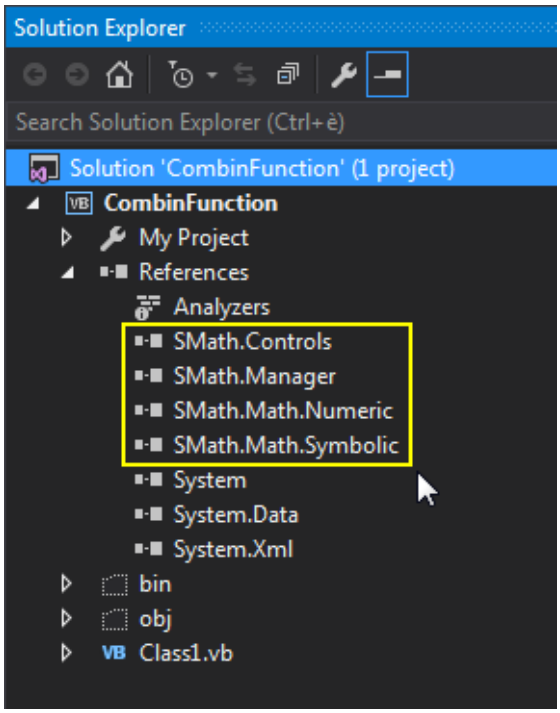


7. Save it, then go back to *Solution Explorer* window, right-click on the project name and then on **Reload Project**. Confirm on the dialog that ask you if you to close all the files, if it is prompted.



If all is gone right, you will see that now the SMath Studio assemblies are loaded in your project (in the *Solution Explorer* expand the **References** item)

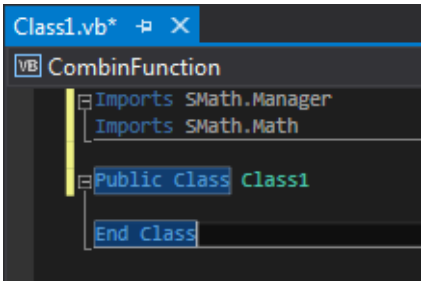
Now everything is ready to start coding!



8. In the *Solution Explorer* double-click on **Class1.vb**

9. In the editing window, above the class definition, type in the following:

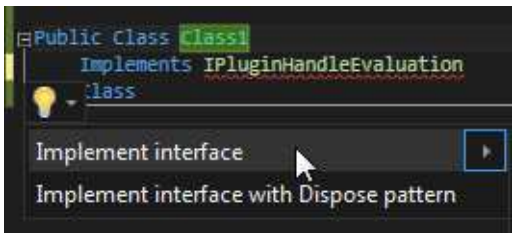
```
Imports SMath.Manager
Imports SMath.Math
```



10. Within the class definition type the following:

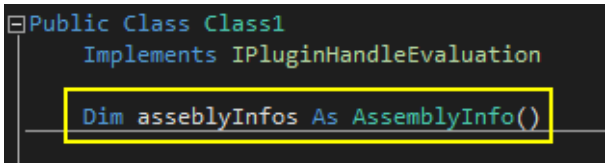
```
Implements IPluginHandleEvaluation
```

then press enter; this will automatically insert an interface (with the interface members) that must be implemented in the class (see endnote 1). You can even click on the light bulb and choose "Implement interface"

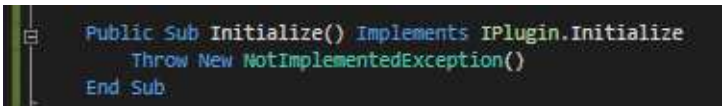


11. Next, type in the following:

```
    Dim asseblyInfos() As AssemblyInfo
```

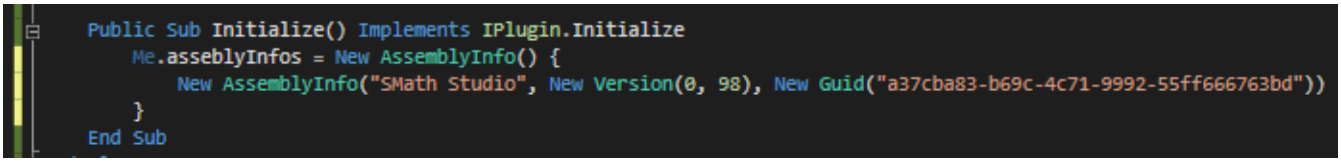


12. Then scroll down the page and find the following subroutine:



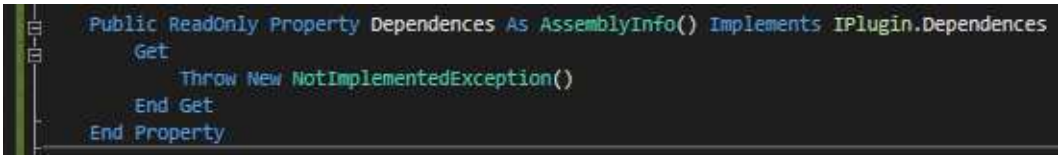
Replace the exception code with this:

```
        Me.asseblyInfos = New AssemblyInfo() {
            New AssemblyInfo("SMath Studio", New Version(0, 98), New
Guid("a37cba83-b69c-4c71-9992-55ff666763bd"))
        }
```



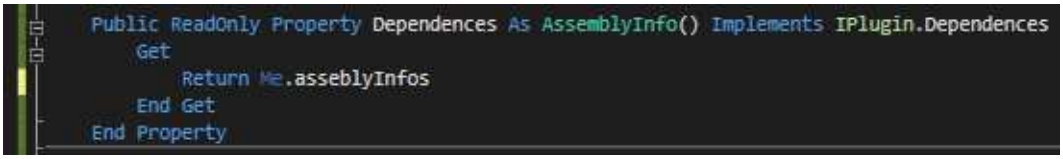
This is required in any plug-in made for SMath Studio.
- The 2nd argument represents the version number of Smath for which you are developing this plug-in. So if you are developing for SMath version 0.98, you insert 98. However, if the version you are targeting is different, enter the appropriate number.
- The 3rd argument will be the same for any plug-in, never change it!

13. Now scroll the code to the following subroutine:



type in the following within the Get block (see endnote 2):

```
            Return Me.asseblyInfos
```



14. Now scroll the code to the following function:

```
    Public Function GetTermsHandled(sessionProfile As SessionProfile) As TermInfo() Impleme
        Throw New NotImplementedException()
    End Function
```

replace the content with the following:

```
        Return New TermInfo() {
            New TermInfo("combin", TermType.Function, "(n, k) - Returns the number of
subsets (combinations) of k elements that can be formed from n elements.",
FunctionSections.Unknown, True, New ArgumentInfo(ArgumentSections.RealNumber), New
ArgumentInfo(ArgumentSections.RealNumber))
        }
```

```
Public Function GetTermsHandled(sessionProfile As SessionProfile) As TermInfo() Implements IPluginHandleEvalua
    Return New TermInfo() {
        New TermInfo("combin", TermType.Function, "(n, k) - Returns the number of subsets (combinations) of k
    }
End Function
```

This allows SMath Studio (and the user) to know several things about your function:
- The 1st argument, "**combin**", is the function name to use inside the worksheets;
- The 2nd argument, **TermType.Function**, is the type of object combin; we'll see it again later;
- The 3rd argument, **"(n, k) - Returns..."**, is the description available in the dynamic assistance;
- The 4th argument, **FunctionSections.Unknown**, is used to group functions by categories (CTRL+E in SS);
- The 5th argument, **true**, is to display the function in the dynamic assistance (use false to hide it).

15. Now scroll to the top and add another interface:

```
IPluginLowLevelEvaluationFast
```

to do it, add a comma after the first interface and type the new one, then implement his members (hit return)

```
Public Class Class1
    Implements IPluginHandleEvaluation, IPluginLowLevelEvaluationFast
```

16. If you scroll down the code, another function is now available:

```
    Public Function TryEvaluateExpression(value As Entry, context As Store, ByRef result As Entry) As Boolean Implemen
        Throw New NotImplementedException()
    End Function
```

type in the following conditional *If* statement:

```
        If value.Type = TermType.Function And value.ArgsCount = 2 And value.Text =
"combin" Then

        End If
```

```
    Public Function TryEvaluateExpression(value As Entry, context As Store, ByRef result As Entry) As Boolean Implem
        If value.Type = TermType.Function And value.ArgsCount = 2 And value.Text = "combin" Then

        End If
    End Function
```

that means "if what is being processing is *my function*, then do something"

17. Now type in the following within the *If* block:

```
            Dim arg1 As Term() = Decision.Preprocessing(value.Items(0),
context).ToTermsList().ToArray()
            Dim arg2 As Term() = Decision.Preprocessing(value.Items(1),
context).ToTermsList().ToArray()
```

```
    Public Function TryEvaluateExpression(value As Entry, context As Store, ByRef result As Entry) As Boolean Implem
        If value.Type = TermType.Function And value.ArgsCount = 2 And value.Text = "combin" Then
            Dim arg1 As Term() = Decision.Preprocessing(value.Items(0), context).ToTermsList().ToArray()
            Dim arg2 As Term() = Decision.Preprocessing(value.Items(1), context).ToTermsList().ToArray()
```

These preprocessing steps are needed to correctly prepare the arguments. This means that all possible substitutions will be performed.

18. Next, type the following:

```
            Dim answer As New List(Of Term)
```

```
    Public Function TryEvaluateExpression(value As Entry, context As Store, ByRef result As Entry) As Boolean Implem
        If value.Type = TermType.Function And value.ArgsCount = 2 And value.Text = "combin" Then
            Dim arg1 As Term() = Decision.Preprocessing(value.Items(0), context).ToTermsList().ToArray()
            Dim arg2 As Term() = Decision.Preprocessing(value.Items(1), context).ToTermsList().ToArray()

            Dim answer As New List(Of Term)
```

This will prepare a container for the answer, made by Terms; these are the low-level units to build math from within the plug-ins. To create the answer, we have to compose an expression array formed in Reverse Polish Notation (see endnote 3). The mathematical expression is:

$$\frac{n!}{(k!)\cdot((n-k)!)}$$

it can be expressed in RPN as:

n ! k ! n k - ! * /

Thus, type in the following lines to compose the list of terms in RPN:

```
            answer.AddRange(arg1)
            answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
            answer.AddRange(arg2)
            answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
            answer.AddRange(arg1)
            answer.AddRange(arg2)
            answer.Add(New Term(Operators.Subtraction, TermType.Operator, 2))
            answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
            answer.Add(New Term(Operators.Multiplication, TermType.Operator, 2))
            answer.Add(New Term(Operators.Division, TermType.Operator, 2))
```

```
⇐ n
⇐ !
⇐ k
⇐ !
⇐ n
⇐ k
⇐ -
⇐ !
⇐ *
⇐ /
```

```
            Dim answer As New List(Of Term)
            answer.AddRange(arg1)
            answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
            answer.AddRange(arg2)
            answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
            answer.AddRange(arg1)
            answer.AddRange(arg2)
            answer.Add(New Term(Operators.Subtraction, TermType.Operator, 2))
            answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
            answer.Add(New Term(Operators.Multiplication, TermType.Operator, 2))
            answer.Add(New Term(Operators.Division, TermType.Operator, 2))
```

19. To finish up the function, type the following right below our *List*:

```
            result = Entry.Create(answer)
            Return True
```

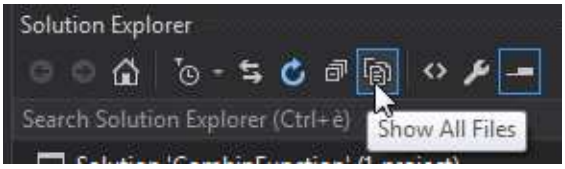
```
            answer.Add(New Term(Operators.Divis

            result = Entry.Create(answer)
            Return True
        End If
```

This will returns the result and that the function we were looking for is found.

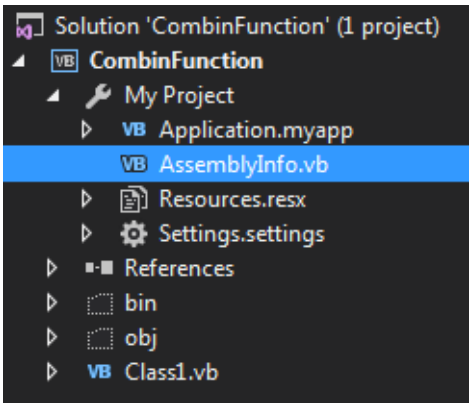
A result is needed even to know if this is not the plug-in that handle the function in evaluation:

```
Public Function TryEvaluateExpression(value As Entry, context As Store, ByRef r
    If value.Type = TermType.Function And value.ArgsCount = 2 And value.Text =
        Dim arg1 As Term() = Decision.Preprocessing(value.Items(0), context).To
        Dim arg2 As Term() = Decision.Preprocessing(value.Items(1), context).To

        Dim answer As New List(Of Term)
        answer.AddRange(arg1)
        answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
        answer.AddRange(arg2)
        answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
        answer.AddRange(arg1)
        answer.AddRange(arg2)
        answer.Add(New Term(Operators.Subtraction, TermType.Operator, 2))
        answer.Add(New Term(Operators.Factorial, TermType.Operator, 1))
        answer.Add(New Term(Operators.Multiplication, TermType.Operator, 2))
        answer.Add(New Term(Operators.Division, TermType.Operator, 2))

        result = Entry.Create(answer)
        Return True
    End If

    Return False
End Function
```

20. The math is done. Now we have to check if the setup of the plug-in is complete; go in the Solution Explorer and select **Show All Files** (if not yet selected).



Navigate to **My Project** ⇒ **AssemblyInfo.vb**, double-click on this file.



21. Now we can edit some attributes (adjust with your owns):

```
' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("Combinations Function")>
<Assembly: AssemblyDescription("Plugin with Combination function realization.")>
<Assembly: AssemblyCompany("Andrey Ivashov")>
<Assembly: AssemblyProduct("Combinations Function")>
<Assembly: AssemblyCopyright("Copyright © 2016 SMath Studio")>
<Assembly: AssemblyTrademark("")>
```

22. There should be a Guid attribute; **if not, you must add it**. Every plug-in **must have a different one**.
It is the identifier of your plug-in, and it is used to save the dependency when you use *combin()* in a worksheet. Remember: *there are many like it, but this one is mine*.

```
'The following GUID is for the ID of the typelib if this project is exposed to COM
<Assembly: Guid("11a097d5-57fc-4ecf-bb38-019d97640221")>
```

IMPORTANT

If it is missing, you can find one guid in the project file (see point 4 above)

CombinFunction.vbproj

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="14.0" DefaultTargets="Build" xmlns="http://schemas
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Micros
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configur
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{C58E81AE-3E63-4F4A-A5C0-8A6143E7E1AE}</ProjectGuid>
    <OutputType>Library</OutputType>
    <RootNamespace>CombinFunction</RootNamespace>
```

23. Last thing here is the version. Add an asterisk for the build and revision numbers of the **AssemblyVersion**, so you will have always a new progressive version every time you will compile the plug-in. *AssemblyFileVersion*, if available, can be safely removed (otherwise you have to update it manually).

```
' Version information for an assembly consists of the following four values:
'
'      Major Version
'      Minor Version
'      Build Number
'      Revision
'
' You can specify all the values or you can default the Build and Revision Numbers
' by using the '*' as shown below:
' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.*")>
```

24. Before testing, we have to open again the project file, as shown in point **4**. Once done, go above the </Project> closing tag in the last line and paste the following:

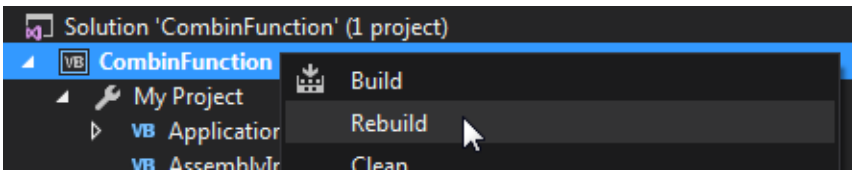
```
  <!-- copy anything from the build path to the SMath Studio extension path -->
  <Target Name="AfterBuild" Condition=" '$(Configuration)' == 'Debug' ">
    <GetAssemblyIdentity AssemblyFiles="$(TargetPath)">
      <Output TaskParameter="Assemblies" ItemName="AssemblyInfo" />
    </GetAssemblyIdentity>
    <GetAssemblyIdentity AssemblyFiles="$(SMathDir)\SMath.Manager.dll">
      <Output TaskParameter="Assemblies" ItemName="ProgramInfo" />
    </GetAssemblyIdentity>
    <PropertyGroup>
      <ProgramVersion>%(ProgramInfo.Version)</ProgramVersion>
      <ConfigFileName>config.$(ProgramVersion.Replace(".", "_")).ini</ConfigFileName>
      <!-- SS portable -->
      <PluginPath Condition=" Exists('$(SMathDir)\portable.version')
">$(SMathDir)\extensions\plugins\$(ProjectGuid.TrimStart("{").TrimEnd("}"))</PluginPath>
      <!-- SS from installer -->
      <PluginPath Condition=" '$(PluginPath)' == ''
">$(APPDATA)\SMath\extensions\plugins\$(ProjectGuid.TrimStart("{").TrimEnd("}"))</Plugin
Path>
    </PropertyGroup>
    <ItemGroup>
      <BuildFiles Include="$(TargetDir)\*.*" />
      <ConfigFileContent Include="%(AssemblyInfo.Version)" />
      <!-- extension status (0: enabled; 2: disabled; 1: removed) -->
      <ConfigFileContent Include="0" />
    </ItemGroup>
    <!-- uncomment line below to keep clean the extension directory -->
    <!-- <RemoveDir Condition="'$(Configuration)' == 'Debug'"
Directories="$(PluginPath)"/> -->
    <Copy SourceFiles="@(BuildFiles)"
DestinationFolder="$(PluginPath)\%(AssemblyInfo.Version)" ContinueOnError="false" />
    <WriteLinesToFile File="$(PluginPath)\$(ConfigFileName)"
Lines="@(ConfigFileContent)" Overwrite="true" />
  </Target>
```

This makes possible to deploy automatically all the build files in the proper directory.

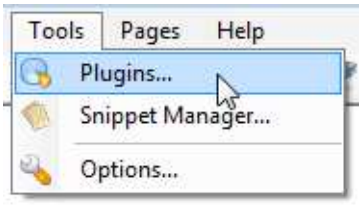
%APPDATA%\Roaming\SMath\extensions\plugins\{GUID}\{version}	for SMath Studio installed
{SMathPath}\Extenions\plugins\{GUID}\{version}	for SMath Studio portable

Save it, then go back to *Solution Explorer* window, right-click on the project name and then on **Reload Project**. Confirm on the dialog that ask you if you to close all the files, if it is prompted.

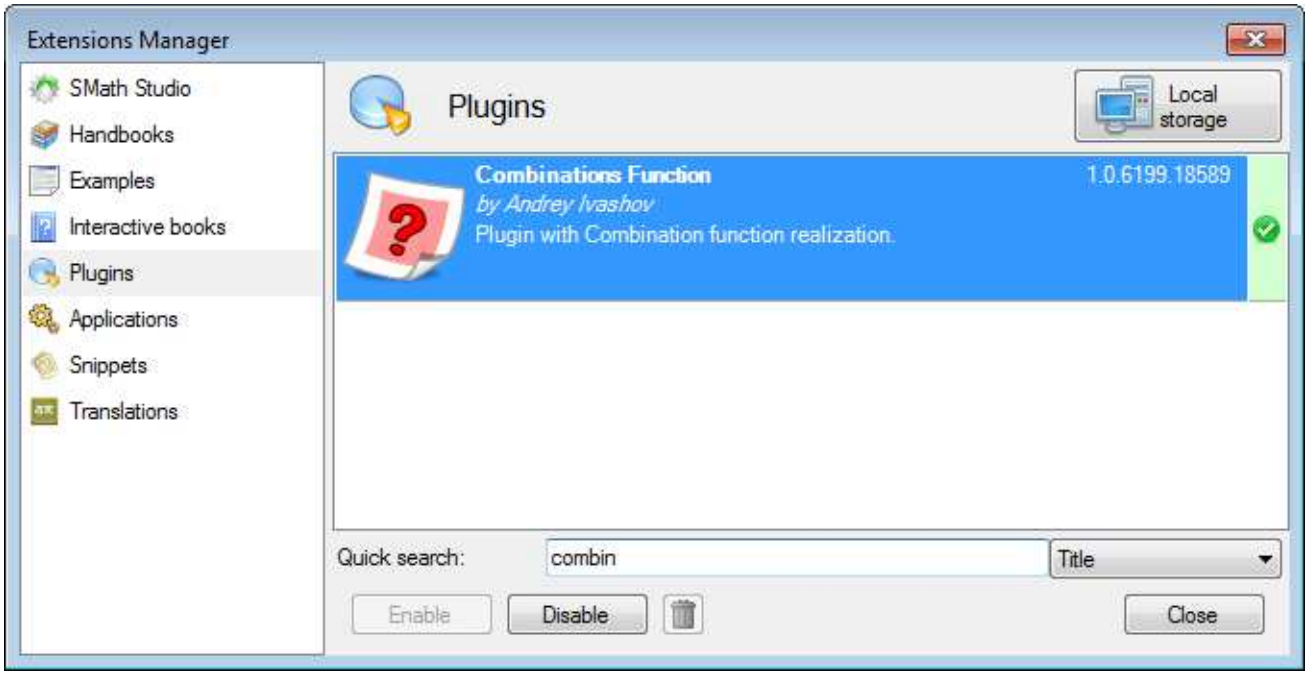
25. Time to test! In the *Solution Explorer*, right-click the solution name and click on **Rebuild**.



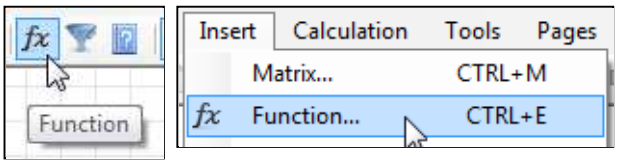
26. Now run *SMath Studio*, then click on **Tools ⇒ Plugins...**

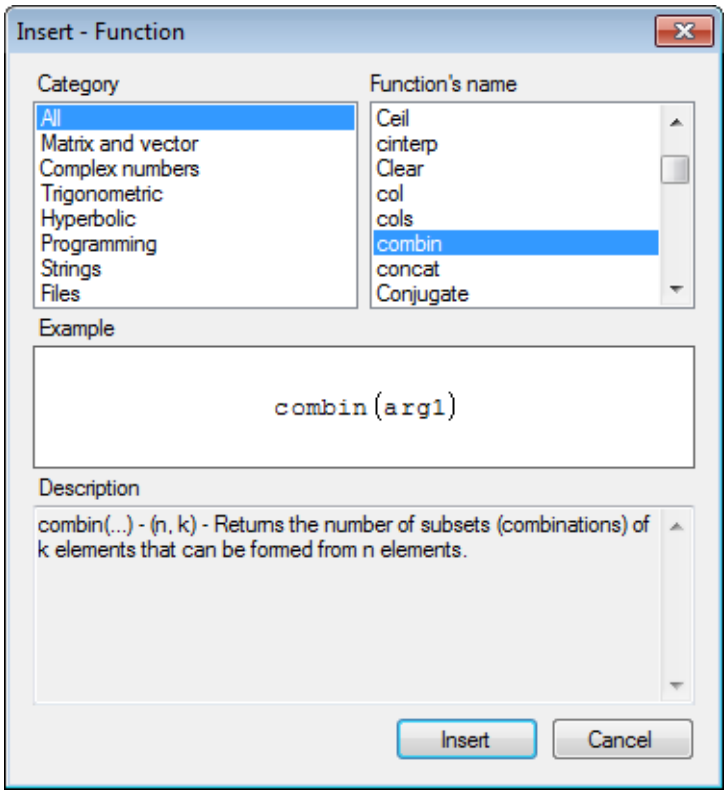


In the *Quick search* field, we search for ***combin***; we'll see that our plugin is loaded and enabled!



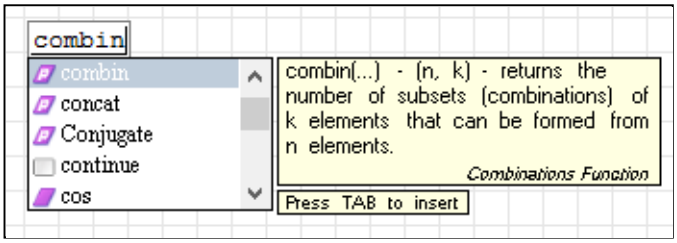
27. Is our function loaded? Go to **Insert** ⇒ **Function...** or click the **Function** symbol on the *Toolbar*.

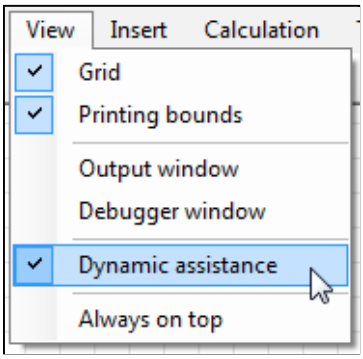




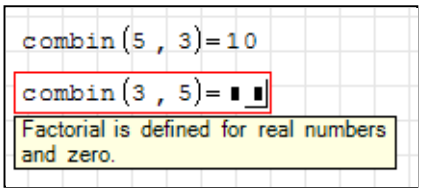
In the *Function's name* list, type **c** and scroll down to find our **combin** function; the description is the one we have defined at point **12**. Since at that point we haven't provided the number of the arguments, it is shown with three points (undefined number of arguments) but only if we will 2 arguments the function will works (because we have defined this behavior at point **16**).

If you type *combin* on the canvas (with Dynamic assistance enabled):





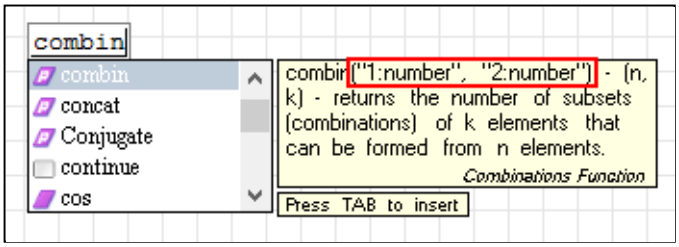
Press TAB and test it. If the result is like in the screenshot below, you have successfully created your first plug-in!



If you go back at point **14**, we can use this to force a 2 arguments function on *TAB* key press

```
        Me.termInfos = New TermInfo() {
            New TermInfo("combin", TermType.Function, "(n, k) - Returns the number of
subsets (combinations) of k elements that can be formed from n elements.",
FunctionSections.Unknown, True, New ArgumentInfo(ArgumentSections.RealNumber), New
ArgumentInfo(ArgumentSections.RealNumber))
        }
```

Once applied, both the number and the type of arguments are shown to the user, and *TAB* will provide a 2 arguments function.



28. However, in the real world, we seldom get by without making mistakes from time-to-time. Let's now show how to debug our plug-in. Typically, you would debug your application before doing steps **24** through **27** that were outlined above. Debugging an application add-in with Visual Studio Community appears to not be as straightforward as in the professional versions of Visual Studio. But below is a workaround that seems to work.

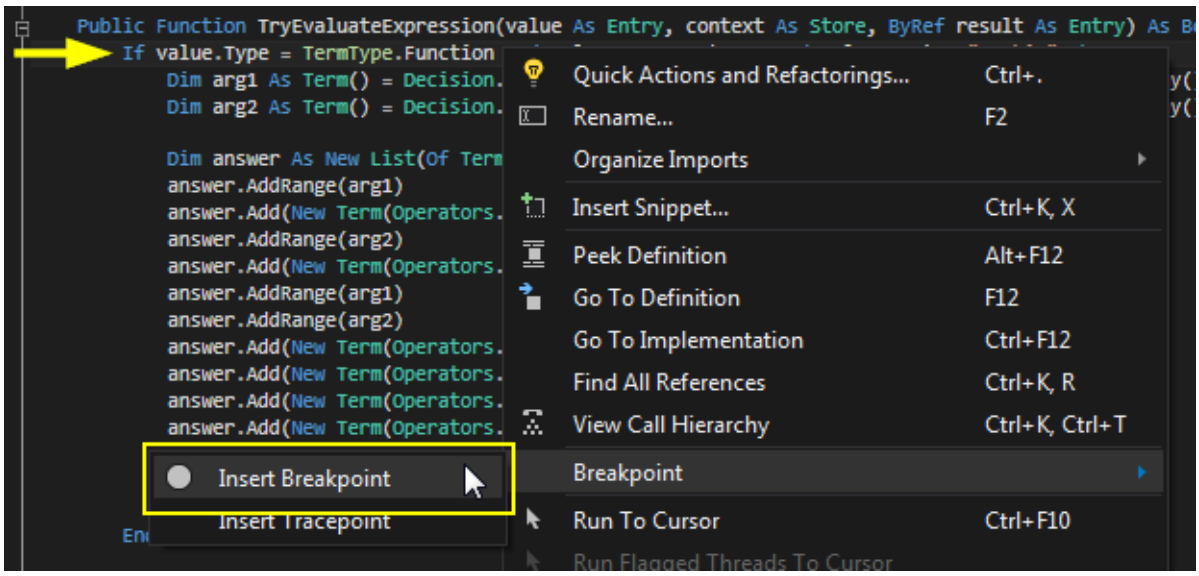
First, we have to open again the project file, as shown in point 4. Once done, under the <PropertyGroup> we have added previously, we can add the following lines:

```
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <StartAction>Program</StartAction>
    <StartProgram>$(SMathDir)\SMathStudio_Desktop.exe</StartProgram>
  </PropertyGroup>
```

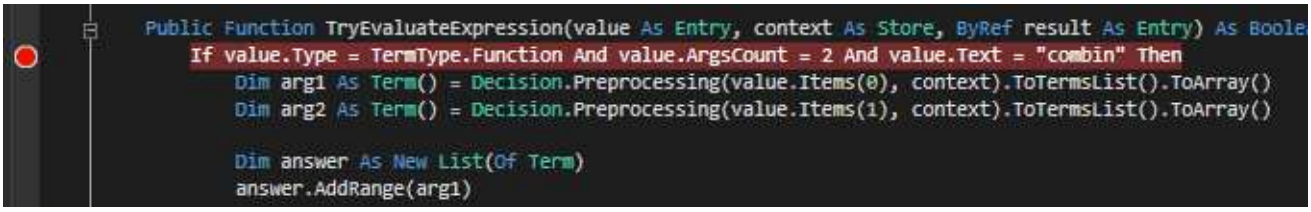
```
<PropertyGroup>
  <!-- Release -> SMath Release Manager -->
  <SMathDir Condition=" '$(SMathDir)' == '' AND '$(Configuration)' == 'Release' "
  <!-- Debug -> development -->
  <SMathDir Condition=" '$(SMathDir)' == '' AND '$(Configuration)' == 'Debug' ">C
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
  <StartAction>Program</StartAction>
  <StartProgram>$(SMathDir)\SMathStudio_Desktop.exe</StartProgram>
</PropertyGroup>
<ItemGroup>
  <Reference Include="SMath.Controls">
    <HintPath>$(SMathDir)\SMath.Controls.dll</HintPath>
    <Private>False</Private>
```

Save it, then go back to *Solution Explorer* window, right-click on the project name and then on **Reload Project**. Confirm on the dialog that ask you if you to close all the files, if it is prompted.

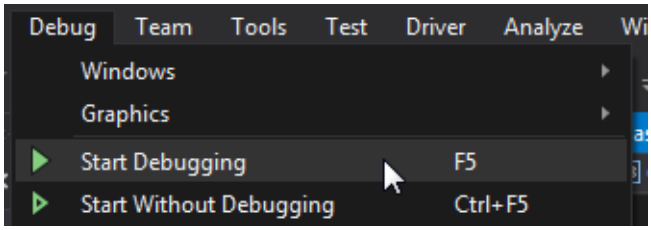
31. Within Visual Basic, set a **breakpoint** at a convenient location. Simply place your cursor in the line at which you wish to set the *breakpoint* and click on **Debug** ⇒ **Toggle Breakpoint** as shown below:



A big red dot will show that the breakpoint is set on the choosen line (the if statement of our function):



32. Start debugging. Click on Debug ⇒ Start Debugging or Start on the Visual Studio toolbar.



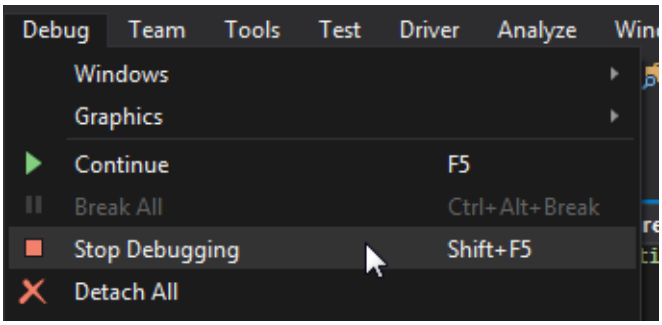
When you do this, Visual Studio will automatically start up Smath Studio and pass the focus to SMath. When this occurs, you must attempt to utilize the plug-in you have created for the purpose of debugging it. In this case, we type in the following:

```
combin(3 , 5)
```

As soon as the "=" is entered, if a *breakpoint* was set, control and screen focus will return to Visual Studio where you can step through the code, watch variable values, and other debugging tasks.

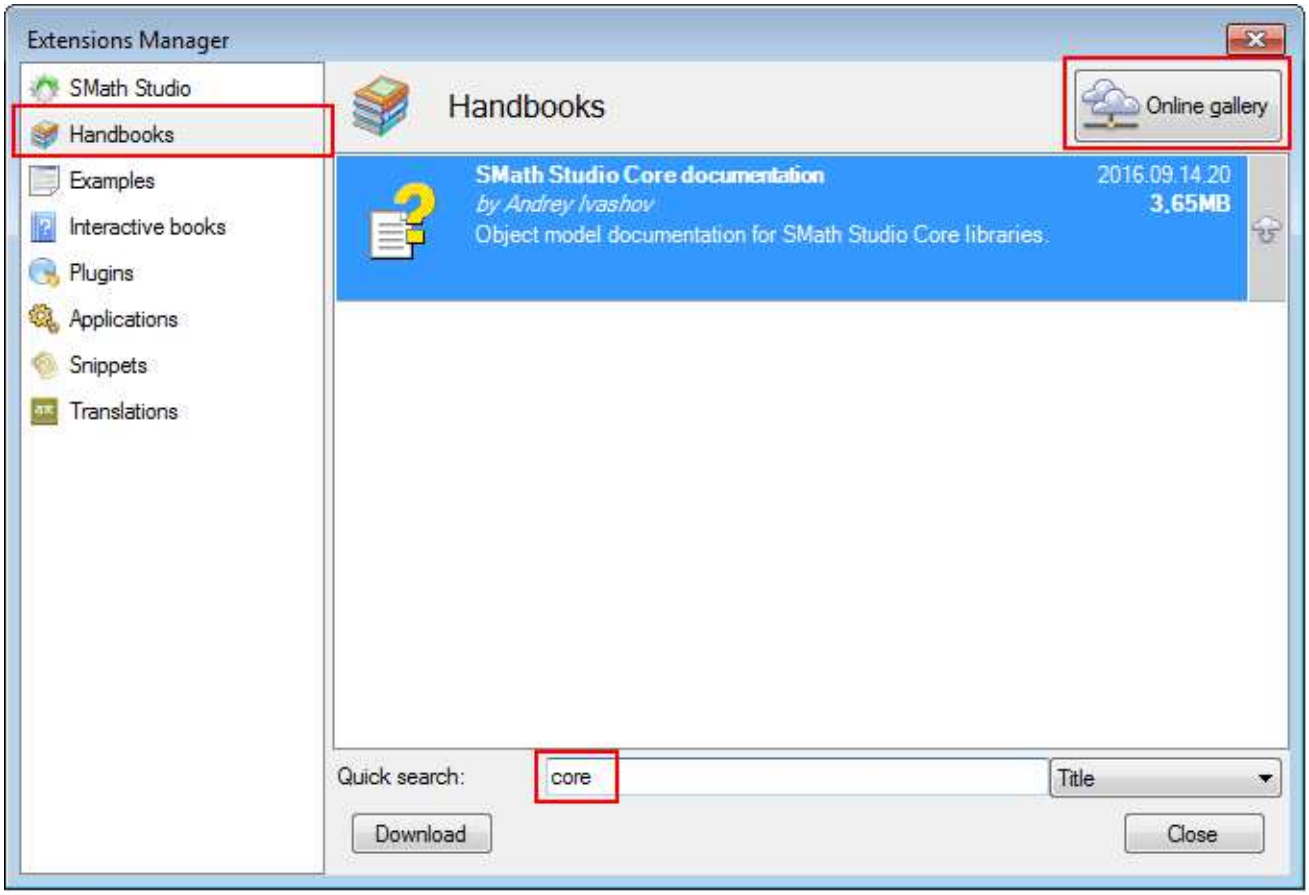
See endnote 5 for some useful links on how to debug your applications within Visual Studio.

33. To stop debugging, click on *Debug* ⇒ *Stop Debugging* as shown below. When you do this, the instance of SMath in which you tested your plug-in will close.



34. Finally, when your plug-in is finished and bug free, you are ready to release it. This essentially involves repeating steps 25 to 28 outlined above, with Relase configuration. The main difference is that the CombinFunction.pdb file does not need to be copied into the “SMath Studio\Plugins” folder.
To know how to relase your plug-in to the community, please visit the following link:
http://en.smath.info/forum/yaf_poststt2399_Extensions-Manager.aspx

Probably you have noticed that the Visual Studio IntelliSense provides hints about methods and properties available for the various namespaces; you can find a list of the featues available within the SMath Studio APIs in his *Extensions Manager*; go to **Tools** ⇒ **Plugins...** ⇒ **Handbooks** then choose **Online gallery** and search the keyword **core**.



Endnotes:
1. Refer to: https://msdn.microsoft.com/en-us/library/7z6hzchx(v=VS.90).aspx
2. Refer to: https://msdn.microsoft.com/en-us/library/38x6w70d(v=VS.90).aspx
3. For explanation of Reverse Polish notation refer to: http://en.wikipedia.org/wiki/Reverse_polish_notation
4. Refer to: https://support.microsoft.com/en-us/kb/865219
5. Here are some useful links about how to debug your applications within Visual Studio
   - Informations on debugging in Visual Studio may be found at:
     http://msdn.microsoft.com/en-us/library/k0k771bt%28v=VS.100%29.aspx
   - Execution Control (stepping through your code):
     http://msdn.microsoft.com/en-us/library/y740d9d3%28v=VS.100%29.aspx
   - Breakpoint Overview:
     http://msdn.microsoft.com/en-us/library/5557y8b4%28v=VS.100%29.aspx
   - Viewing Data in the Debugger:
     http://msdn.microsoft.com/en-us/library/esta7c62%28v=VS.100%29.aspx
   - Edit and Continue:
     http://msdn.microsoft.com/en-us/library/bcew296c%28v=VS.100%29.aspx