# Practice project converting an Add-In to VB.NET with ExcelDna

## Using the Microsoft Visual Basic 2010 Express IDE

Patrick O’Beirne <pob –at- sysmod.com>

Prerequisites: Download ExcelDna and Visual Basic 2010 Express (free). Read the Getting Started projects.

In VB, Tools > Options > Projects and Solutions > check “Show Output window when Build starts”

TOC.xlam is an Excel 2010 add-in that adds a button to the Review tab on the Ribbon, on the far right after the "Changes" group, with a group label “TOC” and a button labelled “Table of Contents”. When you click the button, it inserts a new worksheet into the active workbook with a table of sheet names with hyperlinks to the sheets and some basic information on the sheet contents and size. If you prefer, put it in the Insert tab of the Ribbon.

The topmost folder is named CreateTOC. It contains the example TOC.xlam which is unprotected. Export its two modules TOC.bas and Common.bas

Extract the customUI XML. I use the Custom UI Editor For Microsoft Office or you can simply temporarily rename the file to TOC.xlam.zip, open the zip and extract TOC.xlam.zip\customUI\customUI.xml

In MS VS 2010 Visual Basic

New Project > Class project named CreateTOC.NET

Rename Class1 to CreateTOC.vb

Copy and paste in contents of TOC.bas and Common.bas into separate modules.

Project > Add Reference > Browse tab, to ExcelDNa-0.29\ExcelDna.Integration.dll

In Solution Explorer, press button for “Show All Files”

Select ExcelDna.Integration, make property CopyLocal=False

Project > Add New Item > Text file named CreateTOC.Dna and copy in the text below. I have changed the button ID to CreateTableOfContents rather than InsertTOC as in the addin because I wanted to have both available for testing purposes.

I am using RuntimeVersion 4 because that’s the .Net framework on my PC.

The ExternalLibrary CreateTOC.NET.dll will be created when we build the project.

<DnaLibrary Language="VB" Name="Table of Contents Add-in" RuntimeVersion="v4.0">

<ExternalLibrary Path="CreateTOC.NET.dll" />

<CustomUI>

<!-- Inside here is pasted the contents of the customUI14.xml from the .xlam

Note the <CustomUI> with a capital ‘C’ tag that encloses the <customUI>

with a small ‘c’ tag that is copied from the .xlam file. -->

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">

<ribbon>

<tabs>

<tab idMso="TabReview" >

<group id="customGroup" label="TOC" insertAfterMso="GroupEditingExcel">

<button id="CreateTableOfContents" tag="Insert Table of Contents worksheet"

onAction="RunButtonID" label="&amp;Table of Contents"

screentip="Insert Table of Contents worksheet" size="large"

imageMso="TableOfFiguresInsert" />

</group>

</tab>

</tabs>

</ribbon>

</customUI>

</CustomUI>

</DnaLibrary>

To support the Excel 2007/2010 ribbon, add a module Ribbon.vb with this code:

Imports ExcelDna.Integration.CustomUI

Imports System.Runtime.InteropServices ' needed for <ComVisible(True)>

'class for the Ribbon handler.

'The handler for the button press is defined in the ribbon xml as onAction="RunButtonID"

'which we find in the code module inside the RiskGen.xlam file.

'The ExcelRibbon-derived class must also be marked as ComVisible(True),

' or in the project properties, advanced options, the ComVisible option must be checked.

' (Note that this is not the ‘Register for COM Interop’ option, which must never be used with Excel-DNA.)

<ComVisible(True)> \_

Public Class Ribbon

Inherits ExcelRibbon

'ExcelDna provides a feature to use onAction="RunTagMacro" which will run a VBA sub named in tag="MyVBAMacro"

'I have used the ID for that purpose

Sub RunButtonID(ByVal ctl As IRibbonControl)

‘example: id=”MyMacro” onaction=”RunButtonID”

Application.Run(ctl.Id)

End Sub

Sub RunButtonIDWithTag(ByVal ctl As IRibbonControl)

‘ example: id=”TestTag” onaction=”RunButtonIDWithTag” tag=”Hello”

Application.Run(ctl.Id, ctl.Tag)

End Sub

End Class

Public Module testRibbon

Sub TestTag(ByVal sTag As String)

MsgBox(sTag)

End Sub

End Module

I decided to keep the top level CreateTOC for the addin, the documentation etc and let the VS IDE make the CreateTOC.NET folder and subfolder for the VS Project. Save the project in the CreateTOC folder as CreateTOC.NET. This creates one subfolder named CreateTOC.NET with two files CreateTOC.NET.sln and CreateTOC.NET.suo and another subfolder named CreateTOC.NET. In that subfolder there are the .vbproj and .vbproj.user files (and we will add more) and subfolders bin, My Project, and obj. The “bin” folder has subfolders “Debug” and “Release”. You don’t need to copy any files into Debug or Release, the “Copy file” options in the IDE will determine that, as shown below. Visual Studio automatically chooses the Debug configuration when you choose Start from the Debug menu and the Release configurations when you use the Build menu. You can tailor this from the menu Build > Configuration Manager.

Copy ExcelDna-0.29\Distribution\ExcelDna.xll to the same folder as CreateTOC.DNA and name it "CreateTOC\CreateTOC.NET\CreateTOC.NET\CreateTOC.NET.xll"

Project > Add Existing Item > CreateTOC.NET.xll and set the property Copy to Output Directory to Copy if newer.

To get access to the Excel Object Model:

You may see on some websites ExcelDna.Integration.Excel.Application – this is obsolete; use ExcelDna.Integration.ExcelDnaUtil.Application

Add these two lines to the top of each module:

Imports ExcelDna.Integration ' needed for IExcelAddIn, ExcelDnaUtil

Imports ExcelDna.Integration.ExcelDnaUtil ' for Application object

Now read the code copied in and see what error messages have appeared. The IDE has automatically removed the Set command from object references, this is not used in VB.NET.

Add a module Globals.vb and define Public Constants for all the xl…. enumerations in Excel. I think this is not needed if I use the Excel PIA, eg I can use xlDirection.xlUp but I am taking a minimalist approach first.

Automatic error correction says:

‘Variant’ is no longer a supported type: use the ‘Object’ type instead

Or simply Dim with no type. Ugly but compatible with both VBA and VB.NET.

If there are multiple parameters to a function, and some are declared eg As String, then all must be declared, so use As Object where we had As Range in VBA.

VB.NET does not support default properties like VBA does, e.g. you can't just say

myValue = Application.Range("A1")

but need to explicitly say

myValue = Application.Range("A1").Value

This is one of the most common incompatibilities between VBA and VB.NET.

Don’t use default property of Err, that would get translated to Err(), specify Err.Number. For a better approach someday see

<http://msdn.microsoft.com/en-us/library/ms973849.aspx>

Error Handling in Visual Basic .NET

Now we have the Application object we can resolve all the undefined types like Workbook, worksheet, etc by this edit:

Activeworkbook ->Application.activeworkbook etc

This could also be done other ways:

1) Add helper functions to Globals.vb such as

Function Workbook

Return Application.workbook

2) Use the Excel PIA

3) Use the NetOffice libraries.

The last two are discussed in the background documentation.

Specify ByRef when the variable is meant to be changed and returned. The default for VB.Net is ByVal unlike VBA where it is ByRef.

I also had to do some tidying to ensure a function return is defined for paths not taken, rather than relying on the default value for an uninitialized function variable.

Literal array syntax is different:

'Dim aHeadings As Variant

'aHeadings = Array("Sheet", "Type", "Sheet Tab Name")

Dim aHeadings() As String = {"Sheet", "Type", "Sheet Tab Name”}

Fill in default values for Optional parameters in function headers.

For a while I had a problem with no Ribbon appearing until I fixed two things:

* I had to add <ComVisible(True)> to the Ribbon class (and therefore needed Imports System.Runtime.InteropServices)
* I initially had two controls with the same ID, copied & pasted when testing. The same problem would happen with two group IDs the same. I could have prevented that by pasting the XML into the Custom UI Editor (open any document first) and clicking the Validate button.

To allow me to launch this from the IDE and still debug this in Excel, close VS, and edit CreateTOC.NET.vbproj.user file to read:

<?xml version="1.0" encoding="utf-8"?>

<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">

<StartAction>Program</StartAction>

<StartProgram>C:\Program Files\Microsoft Office\Office14\EXCEL.EXE</StartProgram>

<StartArguments>CreateTOC.NET.xll</StartArguments>

</PropertyGroup>

<PropertyGroup>

<ProjectView>ShowAllFiles</ProjectView>

</PropertyGroup>

</Project>

#END#

Footnotes

Govert Van Drimmelen pointed out “If you are not planning to add helpers for methods like ActiveWorkbook to the Globals.vb, you don't really need to do the Application property and the AutoOpen that sets it. You can always access the Application object via ExcelDnaUtil.Application.”

I had the code below but I can avoid it by having the line

Imports ExcelDna.Integration.ExcelDnaUtil ' for Application object

Module Globals.vb : initialize the Application object

Imports ExcelDna.Integration ' needed for IExcelAddIn

Module Globals

Friend Application As Object

End Module

'Then use the auto open event to set a reference to this variable, from which it can be used through the project. Like this:

Class StartUp

Implements IExcelAddIn

Public Sub Start() Implements IExcelAddIn.AutoOpen

Application = ExcelDna.Integration.ExcelDnaUtil.Application

End Sub

Public Sub Close() Implements IExcelAddIn.AutoClose

'Fires when addin is removed from the addins list but not when excel closes

' - this is to avoid issues caused by the Excel option to cancel out of the close after the event has fired.

MsgBox("Bye bye, from my DNA addin")

End Sub

End Class