# Migrating Excel VBA Add-in to VB.Net, ExcelDNA, NetOffice

This document dated 30 April 2012 builds on the previous documents:

*Getting Started with VB.Net using ExcelDNA.docx*

*Practice project converting XLAM to ExcelDna.docx*

It replaces "ExcelDna addin with Forms.docx" originally 30-Mar-2012

This Excel VBA add-in had 13,000 lines of code and took about two weeks (full-time equivalent) to convert to VB.Net using Visual Studio 2010, ExcelDna 0.29 and NetOffice 1.50. Bear in mind that the supporting libraries are being constantly updated so check for changes in more recent version of ExcelDna and NetOffice. The whole idea of NetOffice is to have version independence, so I don't have to target different versions of Excel - 2007 or 2010, which I would have to do if using the Microsoft office PIAs.

## Fix these first in VBA before doing the migration

The first group are changes that are safe to make in VBA but will make the transition to VB.Net much safer. Fix any issues with Option Base 1 and ByRef first.

### Option Base and Array()

The old code used an array with Option base 1. This is error prone when converting to the 0-based arrays of vb.net so I replaced the array by a class that contained the properties I had been storing in an array. The VBA function Array() can be replicated by defining an Array() function in a GlobalHelpers.vb module:

Function Array(ByVal ParamArray items() As Object) As Array

Return items

End Function

### ByVal and ByRef

VBA defaults to ByRef; VB.Net to ByVal. When passing parameters in VBA, be sure to explicitly specify ByRef so this can be preserved when the module is imported into VS2010. I used to do that for primitive type variables (String, Long, etc) but found I had omitted to do it for Object and Array types. This leads to bugs that can only be detected by executing test cases and comparing output with the VBA output. It would be nice if VS2010 Express could warn us of variables passed byref but changed in the sub. Is this done by tools like Migration Partner and Aivosto Project Analyzer?

### .xl\* Constants

First import the NetOffice Enums module and then prefix enumerated constants with their type, eg XlDirection.xlUp rather than simply xlUp. The prefix can be added in VBA as well which may avoid ambiguities like xlVisible and xlSheetVisible.

Imports NetOffice.ExcelApi.Enums

In the case of VB IDE constants like vbext\* , use

NetOffice.VBIDEApi.Enums.vbext\_ProjectProtection.vbext\_pp\_locked

### .Cells reference

Explicitly specify .Cells in lines like

For Each cell In ActiveSheet.UsedRange.Columns(2).Cells

## Erase

In VBA, the Erase statement clears the value of the elements. The VB.Net Erase statement destroys the elements of an array, and sets the array variable to Nothing.

If you add an array as an item to a collection, it will point to the same array every time unless you re-create the array for each addition to the collection.

You can clear the contents of an array in .Net using:

System.Array.Clear(arr, arr.GetLowerBound(0), arr.Length)

However, any existing pointers to that array (eg if it has been added to a dictionary collection) will now point to the cleared contents. The nearest equivalent to creating a new array is to simply ReDim the array without Preserve.

### Join() function

I had used Join() on an array loaded from a range to concatenate the values into one string. VBA ignored blank cells but vb.net threw an exception because some of the variant array values were Nothing for blank cells. So I created a routine to join only non-blank values.

### IsDate()

This is the same trap as in VBA – it returns True for a string value that looks like a date, so use TypeName(var)="Date" if you want to test for a Date type.

### Evaluate function

This is also a function in .Net which evaluates math expressions, so change all occurrences to Application.Evaluate to get the Excel function.

### Square bracket cell references

Instead of [C2] use .Cells(2,3)

### CutCopyMode

The definition is Public Enum XlCutCopyMode As Integer so the old code

Application.CutCopyMode = False should now be Application.CutCopyMode = 0

### Assigning to cell values

Be explicit about the .Value property, ie not cell = xyz but cell.value = xyz. This avoids bugs when the left hand variable is an Object where you want a variant array of values from the range.

### Initialising variables

There were lots of compiler warnings " Warning 5 Variable 'xx' is used before it has been assigned a value. A null reference exception could result at runtime." The solution is to initialize it eg Dim str as String=vbnullstring

### On Error Resume Next

This practice is known to be error-prone because it suppresses all errors after that line, including ones you did not expect, until reset. It should only be used in one-line functions that are intended to wrap an expected error, eg

' does a sheet of any type exist with this name?

Function SheetExists(ByVal wb As Workbook, ByVal sName As String) As Boolean Dim oSheet As Object

On Error Resume Next

oSheet = wb.Sheets(sName)

Return Err.Number = 0

End Function

### Connection objects

The old .Connection object in Excel pre-2007 has now been replaced by two specific connections: NetOffice.ExcelApi.WorkbookConnection and NetOffice.ExcelApi.ODBCConnection. The connection string is obtained by either

oConn.ODBCConnection.Connection.ToString

oConn.OLEDBConnection.Connection.ToString

For web queries, get the connection from the Querytable in the Ranges (if any):

oConn.Ranges.Item(1).QueryTable.Connection

### Shell & Unzip

Add a reference to Windows\System32\Shell32.dll and use this code

Dim oShell As Shell32.Shell

oShell = New Shell32.Shell ' CreateObject("Shell.Application")

oShell.NameSpace((sFolder)).CopyHere oShell.NameSpace((sZipName)), 16

'16=Respond with "Yes to All"

## The following changes are specific to DotNet

### String$() function

Use StrDup(number,character)

### Format$()

Format$(0.5, "###%") works, "50%"

Format$(1234.5, "###,##0.00") also, "1,234.50"

Format$(Now, "yyyy-mm-dd hh:mm:ss ")

But for time intervals we have to use .ToString

tStart = Now() ……

(Now - tStart).ToString("hh\:mm\:ss")

To convert a time interval to seconds, use CLng((Now - tStart).TotalSeconds)

### IS operator

IS does not work the same as VBA. VBA returns True, Netoffice returns False here:

msgbox( ws.cells(1,1).worksheet is ws.cells(2,2).worksheet )

The workaround to check for two ranges being on the same worksheet is to compare both the workbook & worksheet names.

### Range, Offset, Resize

(This may change with later Netoffice libraries)

We can use Application.Range(strAddress) to get a range from any qualified address.

Since NetOffice 1.5, can use Range.Address rather than the older .get\_Address

A1 Offset(1,1) is A1, get\_Offset(1,1) is B2

A1 Resize(2,3) is C2, get\_Resize(2,3) is A1:C2

### Dictionary Class

The Scripting.Dictionary object can be replaced by a .Net Dictionary, using the syntax

dicWords = New Dictionary(Of String, WordCountItem)

and .Exists becomes .ContainsKey.

VBA: For Each aWordCount In dicWords.Items

Vb.net: For Each pair In dicWords

aWordCount = pair.Value

### Range.Sort

The .Sort method was a headache because every possible parameter must be specified, unlike VBA where they can be omitted. Isolate this into its own sub:

ws.Cells(1, 2 ).CurrentRegion.Sort(header:=XlYesNoGuess.xlYes, \_

key1:=ws.Cells(1, 3 ), order1:=XlSortOrder.xlDescending, \_

key2:=ws.Cells(1, 2 ), order2:=XlSortOrder.xlDescending, \_

key3:=ws.Cells(1, 1 ), order3:=XlSortOrder.xlAscending, \_

dataOption1:=XlSortDataOption.xlSortNormal, \_

dataOption2:=XlSortDataOption.xlSortNormal, \_

dataOption3:=XlSortDataOption.xlSortNormal, \_

matchCase:=False, orderCustom:=Nothing, \_

orientation:=XlSortOrientation.xlSortColumns, \_

sortMethod:=XlSortMethod.xlPinYin, type:=Nothing)

### AutoFilter

Similarly.Autofilter must be fully specified:

ActiveSheet.UsedRange.AutoFilter(field:=1, criteria1:=sCriterion, \_operator:=XlAutoFilterOperator.xlAnd, criteria2:=Nothing, visibleDropDown:=True)

### Workbooks.Open()

This has seventeen parameters. If you want to specify one near the end of the list, like AddToMRU, in VBA you can simply use named parameters, in VB.Net you must specify them all up to that point. You cannot omit them by using two commas in succession. You can pass Nothing for all values except 'format' and 'origin' which must have some real value, eg

Workbooks.Open(filename:=strFilename, updateLinks:=False, readOnly:=False, \_

format:=5, password:=Nothing, writeResPassword:=Nothing, \_

ignoreReadOnlyRecommended:=Nothing, origin:=2, addToMru:=False, \_

converter:=Nothing, corruptLoad:=Nothing, delimiter:=Nothing, \_

editable:=False, local:=Nothing, notify:=Nothing)

'http://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel.workbooks.open(v=office.11).aspx

'format:=5 any value from 1-6 will do if it's not a text file

'origin:=Enums.XlPlatform.xlWindows=2

You cannot specify Format:=Nothing or origin:=Nothing, get:

System.Runtime.InteropServices.COMException (0x800A03EC): Unable to get the Open property of the Workbooks class

### Workbooks.Open() error

error BC30469: Reference to a non-shared member requires an object reference.

This means that you need to qualify 'Workbooks'. Here are two ways:

1. ExcelDnaUtil.Application.Workbooks.Open(). This specifies the Workbooks collection completely.
2. Define a global Application object which you initialise in AutoOpen, and a global Workbooks property (see the GlobalHelpers.vb file) and then you can use simply Workbooks.Open same as in VBA.

### Document Properties

Cast the workbook properties to Netoffice document properties like this:

Dim cdp As NetOffice.OfficeApi.DocumentProperties

cdp = CType(wb.CustomDocumentProperties, NetOffice.OfficeApi.DocumentProperties)

### Getting an Object's Property by Name

As well as CallByName we have Invoker.PropertyGet(workBook, "FileFormat")

### .Characters property gives an error

'Error 16 Class 'NetOffice.ExcelApi.Characters' cannot be indexed because it has no default property.

.Characters(Start:=1, Length:=lPos).Font.ColorIndex = 38

**Mixed .Font.Color returns DBNull**

When there is more than one font colour in a cell, VBA returns 0, VB.Net returns DBNull, which can cause an error comparing DBNull with a number.

### Decimal cell value type

In VBA the possible return values for VarType(cell.value) are 5,6,7,8,10,11. VB.Net adds type 14 (vbDecimal) which is what the Currency type (6) is returned as.

**Controls**

Dim ctl as Control requires Imports System.Windows.Forms and a reference to it.

### DrawingObjects

In VBA, drawing objects are a collection but in Netoffice they are a COMObject

' For Each obj In ws.DrawingObjects gives Error 155 Expression is of type 'LateBindingApi.Core.COMObject', which is not a collection type.

' instead use:

For Each obj In CType(ws, Object).DrawingObjects

### Replace DataObject by Clipboard Class

VB.Net has a Clipboard object with .GetText and .SetText methods which replace the VBA Dataobject.

### Names

To index into the wb.Names collection, use .get\_Name(index)

### SendKeys

Use System.Windows.Forms.SendKeys.Send("{TAB 3}")

­­

### #Error values, CVErr(), IsError()

This seems to have become a mess in .Net. ExcelDna has one set of coding to handle ExcelDna.Integration.ExcelError values passed into, or returned from, UDFs:

<http://groups.google.com/group/exceldna/browse_thread/thread/31e62ad3e2e218b3>

But there are different codes when accessing cell error values in code:

<http://xldennis.wordpress.com/2006/11/29/dealing-with-cverr-values-in-net-part-ii-solutions/>

The CVErr function is gone, the link above has an article by Mike Rosenblum which provides alternatives. The IsError function in VB.Net is nothing to do with the VBA IsError, so again a workaround is needed. In VBA accessing a cell value with an error display of ###### could cause an overflow error 6 in VBA, eg a negative date; this does not happen in VB.Net, it returns the date before 1900, or a Double. I will try the workaround StrDup(Len(cell.Text), "#") = cell.Text

### Restore Excel Status

Place the main code in a Try/Catch that resets event handling, screen updating, etc.

### Unable to get the Paste property of the Worksheet class

This may be caused by a second instance of Excel running.

### Debugging strategies

In VBA Debug.Print had comma and semi-colon separators. In VB.Net Debug.Print(str) takes only one string argument. An alternative is Debug.Write(str) and Debug.WriteLine(str)

Resume or GoTo labels cannot be within a With statement: " Error 69 'GoTo Retry' is not valid because 'Retry' is inside a 'With' statement that does not contain this statement."

With the use of On Error there will be many occurrences of 'first chance' errors, ie those which are trapped by the On Error statement, eg

A first chance exception of type 'System.Runtime.InteropServices.COMException'

You can catch other ones (eg InvalidCastException, NullReferenceException) which should be preventable by enabling debugging on these exceptions but turning off the COMException. In VS2010, click Debug > Exceptions, check 'Thrown' for Common Language Runtime Exceptions, (User-unhandled should be checked too), expand the tree and Uncheck 'System.Runtime.InteropServices.COMException'

See the example GlobalHelper to illustrate the use of the DebugConsole class.

VS2010 has support for breakpoints etc. If you stop debugging (Ctrl+Alt+Break) the Excel instance will be closed and the next time it is restarted it will warn:

*Excel experienced a serious problem with the 'test2 (ribbon helper)' add-in. If you have seen this message multiple times, you should disable this add-in and check to see if an update is available. Do you want to disable this add-in?*

Be sure to click No, otherwise the Ribbon handler will be disabled in all ExcelDna add-ins from that point on.

In VS Express you include this in the .vbproj file so F5 will launch the debugger:

<?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>MY\_ADDIN\_GOES\_HERE.XLL</StartArguments>

</PropertyGroup>

<PropertyGroup>

<ProjectView>ShowAllFiles</ProjectView>

</PropertyGroup>

</Project>

### XML Documents

In VBA you could use oXDoc.SelectSingleNode("//book") but in .Net you need to specify the namespace:

oXDoc = New Xml.XmlDocument

oXDoc.Load(sXMLFileName)

Dim root As XmlElement = oXDoc.DocumentElement

Dim nsmgr As XmlNamespaceManager = New XmlNamespaceManager(oXDoc.NameTable)

nsmgr.AddNamespace("x", root.NamespaceURI) ' x is our temp alias

sNode = "//x:books" ' XPath

oXSectionNode = oXDoc.SelectSingleNode(sNode, nsmgr)

sNode = "//x:books"

oXSectionNodes = oXDoc.SelectNodes(sNode, nsmgr)

'or could do oXDoc.GetElementsByTagName("books")

## UserForms

These had to be redrawn from scratch, there does not seem to be a way to import them. In one case I renamed a form and started getting an error

*The class frmOptions can be designed, but is not the first class in the file.*

I could not fix that so simply deleted and re-created the form again.

In VBA the forms can be referred to by name, eg frmOptions. In VB.Net they have to be instantiated, otherwise you get an error

*Reference to a non-shared member requires an object reference*

http://msdn.microsoft.com/en-us/library/zwwhc0d0(v=vs.80).aspx

says: Avoid adding the Shared keyword

My solution is to name the Class FormCount, instance is Dim frmCount as FormCount, so the same variable name frmCount can be used in the code.

The controls have different properties and methods from VBA.

In VBA I could add listbox or combobox items by .List = array, in vb.net it is

frmOptions.cboClass.Items.AddRange(ary)

The VB.Net ListBox is limited to ONE column of strings, no MultiColumn property. I could use a ListView but that's more complex than I want. So I created a sub to load the items from a range, concatenating the two columns into one string. On the code that used the listbox I then had to parse the item back into its elements, so there is no escaping some added complexity.

VBA .Value of a listbox becomes .SelectedValue

The click event of a listbox is lbxWords\_SelectedIndexChanged

VBA .Value of a checkbox becomes .Checked.

VBA .Caption of a control becomes .Text.

VBA has a .ShowModal property; in vb.net use either the .Show method for non-modal or .ShowDialog for modal.

The events have to have a “handles” clause added to the declaration:

Private Sub cmdOK\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdOK.Click

The parameters can be omitted for simplicity:

Private Sub cmdOK\_Click() Handles cmdOK.Click

In VBA we can access the .Designer property to get lists of controls, that does not seem to be available in .Net.

## Ribbon Interface complications

The ribbon XML is stored in the ProjectName.dna file. For some reason to do with the Ribbon context, it can only see procedures in the Ribbon handling class. If you give it any other sub to call, it will not find it. The only ways to connect them are either:

1. A button has an onAction with the special value of "RunTagMacro" which is an ExcelDna procedure that calls Application.Run on the Tag content.  
   onAction="RunTagMacro" tag="TA\_CountWordsInSelection"  
   Be aware that if the tag name already exists in any other VBA addin loaded before the ExcelDna XLL addin, that sub will be run instead. You have to be sure that the Tag is unique.
2. Or, a button has an onAction with a sub name that is present in the Ribbon handler class.

Here is some code that illustrates both of those:

<DnaLibrary Name="TasksAnalyzer Add-in" RuntimeVersion="v4.0">

<ExternalLibrary Path="TasksAnalyzer.dll" />

<CustomUI>

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

<ribbon startFromScratch="false">

<tabs>

<tab id="Tab\_CCTWAVB" label="Tasks Analyzer" keytip="Q" insertAfterMso="TabAddIns">

<group id="CCTWAVBgroup" label="Task Analyzer (VB)">

<button id="CCTWAVBButton1" onAction="RunTagMacro" tag="TA\_CountInSelection"

label="Analyze Task" screentip="Analyze Tasks Frequency "

size="normal" imageMso="WordCountList" />

<button id="CCTWAVBButton2" onAction="CCTWAVBButton2" tag="TA\_ShowFrmCount"

label="Filter Form" screentip="Show Form to filter analyzed data" size="normal"

imageMso="FilterAdvancedByForm" />

</group>

</tab>

</tabs>

</ribbon>

</customUI>

</CustomUI>

</DnaLibrary>

The Ribbon class is

Imports ExcelDna.Integration.CustomUI ' needed for IRibbonControl

Imports ExcelDna.Integration.ExcelDnaUtil ' for Application object

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

<ComVisible(True)>

Public Class Ribbon

' implement ExcelDna.Integration.CustomUI.ExcelRibbon to get full Ribbon access.

Inherits ExcelRibbon

' Regular Macros can be called from the Ribbon using the onAction='RunTagMacro' helper,

' which calls a macro named in the tag attribute of a button.

' Under the covers, RunTagMacro just calls ExcelDnaUtil.Application.Run(tagName).

'the onAction="MyMacro" attribute will only run methods on the ExcelRibbon-derived class.

'http://groups.google.com/group/exceldna/browse\_thread/thread/60b7fac567c17505

'IDs have to be unique, can’t use same ID for something like "RunIDMacro" more than once.

' This is another way I can think of to pass a parameter from the Ribbon into a macro

Sub RunTagMacroWithID(ByVal ctl As IRibbonControl)

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

' If I try to pass ctl as an object I get

'Cannot run the macro 'TestRibbon'. The macro may not be available in this workbook or all macros may be disabled.

End Sub

Sub CCTWAVBButton2(ByVal ctl As IRibbonControl)

Try

TA\_ShowFrmCount()

Catch ex As Exception

MsgBox(ex.Message, MsgBoxStyle.Exclamation, ex.Source)

End Try

End Sub

End Class

The available keytip letters in Excel are B, C, G, K, Q, S; others are already used by Excel builtin ribbon commands, eg H for Home.

### If the addin menu does not appear in the Ribbon

First paste the XML into the Custom UI Editor (open any document first) and click the Validate button, to check for bad XML or duplicate IDs.

In the ExcelDna forum Serge says: it turns out that in Excel 2007 (may apply to other versions) if a Ribbon at some point causes Excel to crash, all custom Ribbons will be disabled in the future. To re-enable custom ribbons, I had to go to Excel Options => Add-Ins => Manage: Disabled Items Go... => Click on the disabled ExcelDna addin and click Enable.

## Global Helper Functions

Prefix enumerated constants with their type, eg XlDirection.xlUp rather than simply xlUp (which is Global in VBA). This can be done in the VBA as well, so it is compatible both ways. In the absence of that, the change has to be done in the VB.Net IDE after copying the code from the VBA addin. One way of simplifying the amount of editing is to define a GlobalHelper.vb module that provides some compatibility code for constants and properties like Application.Selection.

It is also necessary to change some properties to the ExcelDna get\_ names:

.Range 🡪 .get\_Range

.Address 🡪 .get\_Address

.Offset 🡪 .get\_Offset

.Resize 🡪 .get\_Resize

VBA accepts a variable number of parameters, eg for .Offset we can specify only a row offset and the column offset defaults to 0. They must be specified in ExcelDna so after the search/replace for the .get\_ names, some parameters may have to be completed. The only one I found irritating was .get\_Address which needs five parameters so I decided to create a function Range\_Address which handles the optional parameters and will also convert 0/1 to False/True as required when that kind of lazy shortcut was taken when writing the VBA code.

To do a search and replace of <expression>.Address(…

Visual Studio regular expressions: Find and Replace

'{:a+}\.address

'range\_address(\1)

Notepad++ regular expressions: Search for any text followed by a space or bracket then any text followed by .address( and replace it by the first two subexpressions then Range\_Address( then the text just before .address, which should be the range object reference:

(.\*)([ \(])(.\*?)(\.address\()

Replace with:

\1\2 Range\_Address(\3,

The code is in the GlobalHelper.vb file on the web site:

<http://www.sysmod.com/GlobalHelper.vb>

It defines functions and properties to support the VBA Array(), IsEmpty(), IsNull(), IsObject(), Round(), Selection, ActiveCell, ActiveSheet, ActiveChart, ActiveWindow, ActiveWorkbook, and the Workbooks collection.

### "Reference to a non-shared member requires an object reference"

This error from "Workbooks.Open(…" illustrates a typical need for the helpers such as Workbooks. Firstly I must create an OBJECT called "Application" Then EITHER I change all "Workbooks" in the VBA code to an explicit reference to Application.Workbooks which requires editing the code

OR I create a public module with a Property Workbooks. That is Govert's solution, and results in a Global Helper with lots of ReadOnly Properties like:

ReadOnly Property Workbooks As Workbooks

Get

Return Application.Workbooks

End Get

End Property

1) Do this in a Public Module GlobalHelper

Property Application As Netoffice.ExcelApi.Application

2) Do this in a Public Class DnaAddIn

Public Sub AutoOpen() Implements IExcelAddIn.AutoOpen

Application = new Netoffice.ExcelApi.Application(nothing, ExcelDnaUtil.Application)

I need Netoffice.ExcelApi. or I get

'error BC30561: 'Application' is ambiguous, imported from the namespaces or types 'System.Windows.Forms, NetOffice.ExcelApi'.

## Compatibility with VBA code that references ThisWorkbook

I typically store some setup parameters in properties and worksheets in the xlam file. Instead of worksheets, in future projects I will have to use configuration files. For properties, I could replace them by constants but for an exercise I defined a ThisWorkbook class in the AddInMain.vb module and created properties as follows:

'Create an ExcelAddIn-derived class with AutoOpen and AutoClose,

'and add a module called AddInMain to hold the Application object reference:

Imports LateBindingApi.Core

Imports NetOffice.ExcelApi

Imports ExcelDna.Integration

' This class is implemented only to allow us to initialize NetOffice

' We hook up a public field in the Module AddInMain

' that will be usable anywhere in the project.

Public Class AddIn

Implements IExcelAddIn

Public Sub AutoOpen() Implements IExcelAddIn.AutoOpen

' must initialise here because XlCall cannot be used from Ribbon context, only in a macro context

ThisWorkbook.Name = System.IO.Path.GetFileName(XlCall.Excel(XlCall.xlGetName))

ThisWorkbook.Path = System.IO.Path.GetDirectoryName(XlCall.Excel(XlCall.xlGetName))

Factory.Initialize()

' Set a public field in a module, so that Application will be available everywhere.

Application = New Application(Nothing, ExcelDnaUtil.Application)

End Sub

Public Sub AutoClose() Implements IExcelAddIn.AutoClose

End Sub

End Class

Public Class ThisWorkbook

'Shared means we don't need to instantiate ThisWorkbook to call these

Shared Property Title As String = DnaLibrary.CurrentLibrary.Name

Shared Property Name As String = "ThisWorkbook"

Shared Property Path As String = "."

Shared ReadOnly Property Worksheets As Object

Get

MsgBox("No Worksheets in ThisWorkbook")

Return Nothing

End Get

End Property

Shared Function IsAddin() As Boolean

Return True ' for debugging

End Function

End Class

### Unit Testing using NotePad++ as the code editor

To simplify testing isolated pieces of code, I used NotePad++ to edit a .Dna file and copied ExcelDna.xll to the test filename.xll. Then to run the test, I saved a command in Notepad++ assigned to Ctrl+F5:

"C:\Program Files\Microsoft Office\Office14\EXCEL.EXE" "$(CURRENT\_DIRECTORY)\$(NAME\_PART).XLL"

Here is a sample test .Dna file:

<DnaLibrary Language="VB" RuntimeVersion="v4.0" >

<!-- Notepad++

Run "C:\Program Files\Microsoft Office\Office14\EXCEL.EXE" "$(CURRENT\_DIRECTORY)\$(NAME\_PART).XLL"

-->

<Reference Path="LateBindingApi.Core.dll" />

<Reference Path="System.Windows.Forms.dll" />

<Reference Path="ExcelApi.dll" />

<![CDATA[

' lines above this point are not counted in the compiler error messages

' Add 10 to reported error line number to get line in this file

Imports NetOffice.ExcelApi ' Application, Workbook etc.

'Cannot use Imports Excel=, causes Dim as Workbook to fail

Imports LateBindingApi.Core ' DebugConsole

Imports System.Windows.Forms ' Clipboard

Public Class DnaAddIn

Implements IExcelAddIn ' ExcelDna.Integration included implicitly from ExcelDna.xll copied to $(NAME\_PART).xll

Public Sub AutoOpen() Implements IExcelAddIn.AutoOpen

LateBindingApi.Core.Factory.Initialize() ' Initialize NetOffice

'error BC30561: 'Application' is ambiguous, imported from the namespaces or types 'System.Windows.Forms, NetOffice.ExcelApi'.

Application = new NetOffice.ExcelApi.Application(nothing, ExcelDnaUtil.Application)

'http://netoffice.codeplex.com/wikipage?title=Misc01

' activate the LateBindingApi.Core.DebugConsole. the default value is: ConsoleMode.None

'DebugConsole.Mode = ConsoleMode.MemoryList

DebugConsole.FileName="F:\DOCS\NetOfficeTest\DebugConsole.log"

DebugConsole.Mode = ConsoleMode.LogFile

TestASub()

End Sub

Public Sub AutoClose() Implements IExcelAddIn.AutoClose ' Must be declared even if not used

End Sub

End Class

' This module contains <global> shortcuts to the Application members.

Public Module GlobalHelper

Property Application As NetOffice.ExcelApi.Application ' linked to ExcelDnaUtil.Application in AutoOpen

ReadOnly Property ActiveWorkbook As Workbook

Get

Return Application.ActiveWorkbook

End Get

End Property

'…etc… see GlobalHelper.vb for more

End Module

Public Module MainProcs

Sub TestASub()

try

' this works thanks to the Global Helper class Property "Workbooks"

dim wb as Workbook=Workbooks.Add()

dim ws as Worksheet=wb.worksheets(1)

' can use .Range or .get\_Range. use Application.Range for refs to other workbooks or sheets

msgbox(Application.range("Sheet2!A2:C3").Address(false,false,Enums.xlReferenceStyle.xlA1,true) & vbcrlf & ws.range(ws.cells(2,1),ws.cells(3,3)).address(false,false,Enums.xlReferenceStyle.xlA1,true))

wb.Close(SaveChanges:=False)

catch Ex as Exception

msgbox(Ex.toString ,vbExclamation,"Error")

Console.WriteLine(Ex.toString)

end try

Application.Quit

End Sub

End Module

]]>

</DnaLibrary>

#END#

Patrick O'Beirne

Mail3 at sysmod.com

Web: <http://www.sysmod.com>

Blog: <http://sysmod.wordpress.com>

LinkedIn: <http://www.linkedin.com/in/patrickobeirne>

Skype: sysmodltd