Creating An XLA Add-In For Excel

This page describes how to write an XLA Add-In For Excel.  
ShortFadeBar

Introduction

An XLA Add-In is a certain type of workbook that provides custom functions and/or tools that extend the basic functionality of Excel. An Add-In may contain User Defined Functions (UDFs, see [Writing Your Own Functions In VBA](http://www.cpearson.com/excel/WritingFunctionsInVBA.aspx)) that provide calculation functions specific to your own area of interest or business, functions that extend beyond the normal calculation capability of Excel. An Add-In can also provide tools to manipulate the data in a workbook. Indeed, an Add-In can be written to do nearly anything you want. If it can be done manually, it can be automated with an XLA Add-In.

This page describes XLA Add-Ins written in VBA. For information about other types and uses of Add-Ins, see:

* [Automation Add-Ins As Function Libraries With VB6](http://www.cpearson.com/excel/AutomationAddIns.aspx)
* [COM Add-Ins In Excel 2007](http://www.cpearson.com/excel/COMAddIn2007.aspx)
* [Creating A COM Add-In With Visual Basic 6](http://www.cpearson.com/excel/creatingcomaddin.aspx)
* [Creating A Function Library In VB.NET](http://www.cpearson.com/excel/CreatingNETFunctionLib.aspx)
* [Installing COM Add-Ins](http://www.cpearson.com/excel/COMAddInsSecurity.aspx)
* [Writing User Defined Functions In VBA](http://www.cpearson.com/excel/WritingFunctionsInVBA.aspx)

You can [download the sample XLA file](http://www.cpearson.com/Zips/TestAddIn.zip) used as an example in this article.  
  
An XLA Add-In differs from a normal XLS workbook in the following areas:

* An Add-In has a file extension of .xla rather than .xls;
* The IsAddIn property of the Workbook object is True;
* An Add-In is not visible. Any user interaction must be provided by command bars and/or menu items;
* An Add-In is not returned in the enumeration of the Workbooks object;
* User Defined Functions in an Add-In may be called directly from worksheet cells without the workbook name prefix.

SectionBreak

Creating The Add-In File

Create a new, empty workbook and select *Save As* from the *File* menu. Enter the name of the Add-In workbook and choose *Microsoft Office Excel Add-In (\*.xla)* near the bottom of the *Save as type* dropdown box. Excel will change the folder to the standard library path. This is the folder specified by the property Application.UserLibraryPath. The exact location depends on your version of Excel and your version of Windows. You can save the file to that folder or you can navigate in the Open dialog to any folder you wish. (See *Where To Install The Add-In* below.) Click *Save* and then press ALT F11 to open the VBA Editor.

If it appears that Excel didn't properly save the add-in, go to the *File* menu in Excel (not VBA) and choose *Open*. Navigate to the folder in which you saved the Add-In and open the Add-In file. Go back to the VBA Editor. The XLA project should now be visible in the Project Explorer windows (CTRL R if the window is not visible). This is a known problem in Excel.

The XLA file will not be visible in Excel because Add-Ins are never visible. Everything you need to do is done in the VBA Editor. If your XLA needs to provide any user interaction (that is, it is more than just a library of functions) you need to create command bar items and/or menu items for the user. In general, you should use VBA code to create the command controls when the Add-In is loaded (by putting the code in the Workbook\_Open event procedure in the ThisWorkbook code module) and then delete those controls when the Add-In is unloaded (by putting the code in the Workbook\_BeforeClose event procedure in the ThisWorkbook code module). Sample code for a complete ThisWorkbook code module is on the [sample XLA code page](http://www.cpearson.com/excel/createaddinsamplecode.aspx).

Next, add a regular code module to the VBA Project. In that module, insert the code procedures that you assigned to the OnAction property of the menu item. Some very simple code is shown on the [sample XLA code page](http://www.cpearson.com/excel/createaddinsamplecode.aspx). Of course, this module can contain other code as well. What else goes in the add-in is up to you.

SectionBreak

Where To Install The Add-In

You can store the XLA file in any folder you want. The default location for add-ins is the folder named by the Application.UserLibraryPath property. The value will depend on your version of Excel and your version of Windows. On my Windows Vista Ultimate machine, that path, in both Excel 2003 and 2007, is  
  
C:\Users\Pearson\AppData\Roaming\Microsoft\AddIns  
  
If you save the XLA file in the Application.UserLibraryPath folder, Excel will automatically add the Add-In to the list of available Add-Ins displayed in the Add-Ins dialog box. If you save the XLA to another folder, you will need to click the *Browse* button in the Add-Ins dialog and navigate to your XLA file. In either case, you will need to check the box next to your add-in to open and load the Add-In.

SectionBreak

Properties Of The Add-In

You should change the file properties to display the proper text in the Add-Ins dialog box. In the Immediate window (CTRL G to display it if it is not visible) in the VBA Editor, change the IsAddIn property to False so that the XLA workbook will be visible in Excel. Enter the following in the Immediate window and press ENTER:  
  
ThisWorkbook.IsAddIn = False  
  
Now, go back to Excel and choose *Properties* from the *File* menu. Change the *Title* property to the text that you want to display in the *Add-Ins available* list box. Change the *Comments* property to the text that you want to display in the Add-Ins dialog comment box. Once you have changed the file properties, you need to change the IsAddIn property back to True. Enter the same command in the Immediate window as you entered earlier but change False to True. Finally, save the file.

SectionBreak

Calling Add-In Functions In VBA

If you want to make the functionality of your Add-In avaiable to VBA code, you need to change two properties of the VBA Project. With your XLA project active in the VBA Editor, go to the *Tools* menu and choose *VBAProject Properties...* and change the *Project Name* to something meaningful and change the *Project Description* to some short meaningful description of the Add-In. If desired, you can choose the *Protection* tab and password protect the VBA code if you don't want others to access the code.

CAUTION: The security implemented by password protecting a VBA Project is extremely weak. Password breakers are widely available. I use Passware's VBAKey program and it can break a VBA password in a matter of seconds. You should think of the password protection as a means to prevent an innocent user from mistakenly breaking something in the code. The security is not strong enough to protect proprietary code or code with intellectual property value. If you need that level of protection, you should be using a COM Add-In, not an XLA Add-In.

Once you have set these properties, other VBA Projects can reference your XLA by setting a reference to the project using the name you provided for the *Project Name*. To set a reference from a workbook to the add-in, open the file that is to use the reference, go to the *Tools* menu and choose *References*. In the dialog that displays, select your add-in project in the list and check that item.

You can [download the sample XLA file](http://www.cpearson.com/Zips/TestAddIn.zip) used as an example in this article.  
See also [Installing An XLA](http://www.cpearson.com/excel/InstallingAnXLA.aspx) for other factors involved in deploying and installing XLA Add-Ins on other machines.

This page last updated: 23-January-2011

Sample Code For An XLA Add-In

This page contains sample code for an XLA Add-In.  
ShortFadeBar

This page contains sample code for the XLA Add-In described at [Creating An XLA Add-In](http://www.cpearson.com/excel/CreateAddIn.aspx).

SectionBreak

ThisWorkbook Code Module

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' START ThisWorkbook Code Module

' Created By Chip Pearson, chip@cpearson.com

' Sample code for Creating An Add-In at http://www.cpearson.com/Excel/CreateAddIn.aspx

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

Private Const C\_TAG = "ChipAddIn" ' C\_TAG should be a string unique to this add-in.

Private Const C\_TOOLS\_MENU\_ID As Long = 30007&

Private Sub Workbook\_Open()

'''''''''''''''''''''''''''''''''''''''''''''''

' Workbook\_Open

' Create a submenu on the Tools menu. The

' submenu has two controls on it.

'''''''''''''''''''''''''''''''''''''''''''''''

Dim ToolsMenu As Office.CommandBarControl

Dim ToolsMenuItem As Office.CommandBarControl

Dim ToolsMenuControl As Office.CommandBarControl

'''''''''''''''''''''''''''''''''''''''''''''''

' First delete any of our controls that

' may not have been properly deleted previously.

'''''''''''''''''''''''''''''''''''''''''''''''

DeleteControls

''''''''''''''''''''''''''''''''''''''''''''''

' Get a reference to the Tools menu.

''''''''''''''''''''''''''''''''''''''''''''''

Set ToolsMenu = Application.CommandBars.FindControl(ID:=C\_TOOLS\_MENU\_ID)

If ToolsMenu Is Nothing Then

MsgBox "Unable to access Tools menu.", vbOKOnly

Exit Sub

End If

''''''''''''''''''''''''''''''''''''''''''''''

' Create a item on the Tools menu.

''''''''''''''''''''''''''''''''''''''''''''''

Set ToolsMenuItem = ToolsMenu.Controls.Add(Type:=msoControlPopup, temporary:=True)

If ToolsMenuItem Is Nothing Then

MsgBox "Unable to add item to the Tools menu.", vbOKOnly

Exit Sub

End If

With ToolsMenuItem

.Caption = "&Menu Item"

.BeginGroup = True

.Tag = C\_TAG

End With

''''''''''''''''''''''''''''''''''''''''''''''

' Create the first control on the new item

' in the Tools menu.

''''''''''''''''''''''''''''''''''''''''''''''

Set ToolsMenuControl = ToolsMenuItem.Controls.Add(Type:=msoControlButton, temporary:=True)

If ToolsMenuControl Is Nothing Then

MsgBox "Unable to add item to Tools menu item.", vbOKOnly

Exit Sub

End If

With ToolsMenuControl

''''''''''''''''''''''''''''''''''''

' Set the display caption and the

' procedure to run when clicked.

''''''''''''''''''''''''''''''''''''

.Caption = "Click Me &One"

.OnAction = "'" & ThisWorkbook.Name & "'!MacroToRunOne"

.Tag = C\_TAG

End With

''''''''''''''''''''''''''''''''''''''''''''''

' Create the first control on the new item

' in the Tools menu.

''''''''''''''''''''''''''''''''''''''''''''''

Set ToolsMenuControl = ToolsMenuItem.Controls.Add(Type:=msoControlButton, temporary:=True)

If ToolsMenuControl Is Nothing Then

MsgBox "Unable to add item to Tools menu item.", vbOKOnly

Exit Sub

End If

With ToolsMenuControl

''''''''''''''''''''''''''''''''''''

' Set the display caption and the

' procedure to run when clicked.

''''''''''''''''''''''''''''''''''''

.Caption = "Click Me &Two"

.OnAction = "'" & ThisWorkbook.Name & "'!MacroToRunTwo"

.Tag = C\_TAG

End With

End Sub

Private Sub Workbook\_BeforeClose(Cancel As Boolean)

''''''''''''''''''''''''''''''''''''''''''''''''''''

' Workbook\_BeforeClose

' Before closing the add-in, clean up our controls.

''''''''''''''''''''''''''''''''''''''''''''''''''''

DeleteControls

End Sub

Private Sub DeleteControls()

''''''''''''''''''''''''''''''''''''

' Delete controls whose Tag is

' equal to C\_TAG.

''''''''''''''''''''''''''''''''''''

Dim Ctrl As Office.CommandBarControl

On Error Resume Next

Set Ctrl = Application.CommandBars.FindControl(Tag:=C\_TAG)

Do Until Ctrl Is Nothing

Ctrl.Delete

Set Ctrl = Application.CommandBars.FindControl(Tag:=C\_TAG)

Loop

End Sub

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' END ThisWorkbook Code Module

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

SectionBreak

Module1 Code Module

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' START Module1 Code Module

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

Sub MacroToRunOne()

Dim S As String

S = "Hello World From One:" & vbCrLf & \_

"This Add-In File Name: " & ThisWorkbook.FullName

MsgBox S

End Sub

Sub MacroToRunTwo()

Dim S As String

S = "Hello World From Two:" & vbCrLf & \_

"This Add-In File Name: " & ThisWorkbook.FullName

MsgBox S

End Sub

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' END Module1 Code Module

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

This page last updated: 8-October-2007

Installing An Add-In

This page describes how to install an XLA Add-In for Excel.  
ShortFadeBar

Adding A New Add-In

If you have downloaded or otherwise acquired an XLA Add-In, you must install it so that Excel is aware of its existence and can load the add-in as necessary to access the functions and tools provided by the Add-In.

In general usage when speaking of XLA Add-Ins, the term *install* means to add the Add-In to Excel's list of known add-ins. Typically this is done only once, when you first acquire an Add-In and add it to your machine. The term *load* means to actually open the Add-In and make its functionality available. *Loading* an Add-In is essentailly the same as opening a workbook. The file is opened and loaded into memory. Confusingly, VBA uses the term *Installed* to mean what is generally meant by *loaded*.

SectionBreak

Installing A New Add-In

In a typical Office/Excel installation, Excel will expect Add-Ins to be placed in either of two locations. These locations vary depending on the version of Excel and the version of Windows, but are accessible via the Application.LibraryPath and Application.UserLibraryPath properties. On my current machine, running Windows Vista Ultimate and Office 2007 Ultimate, these paths are:

LibraryPath: C:\Program Files\Office 2007\Office12\LIBRARY

UserLibraryPath: C:\Users\Pearson\AppData\Roaming\Microsoft\AddIns

To see the values of these properties on your machine, run the following code:

Sub ShowLibraryPaths()

MsgBox "Library Path: " & Application.LibraryPath & vbCrLf & \_

"User Library Path: " & Application.UserLibraryPath, vbOKOnly

End Sub

It is not required, however, that an Add-In reside in one of those folders. An Add-In may be placed anywhere in any folder on your machine. Once you have placed the file in the appropriate folder, go to the *Tools* menu and choose *Add-Ins...*. (In Excel 2007, click the Office button, choose *Excel Options* then select the *Add Ins* screen, choose *Excel Add-Ins* in the *Manage* input box and click *Go*.) In the Add-Ins dialog, click the *Browse* button. This will open a familiar Open File dialog. Navigate to the folder in which you saved the XLA Add-In file, select that file, and click *Open*. This will install the Add-In. Generally, you need to do this only one time, the first time you use the Add-In.

To install an Add-In using VBA code, use code like the following:

Sub InstallAddIn()

Dim AI As Excel.AddIn

Set AI = Application.AddIns.Add(Filename:="C:\MyAddIn.xla")

AI.Installed = True

End Sub

SectionBreak

Loading An Existing Add-In

After an Add-In has been installed, it must be loaded in order to access its functions and tools. Open the Add-Ins dialog as described above and locate the Add-In in the list. Note that Add-Ins are listed in the dialog box by the text that was assigned for the Title property of the workbook when it was created, not by disk file name. Select the Add-In and check the box next to it and click OK. To unload an Add-In, do the same thing but uncheck the box next to the Add-In item.

To load or unload an Add-In using VBA code, use code like the following:

Sub InstallAddIn()

''''''''''''''''''''''''''''''''''''

' Installed = True to load Add In,

' Installed = False to unload Add In

''''''''''''''''''''''''''''''''''''

Application.AddIns("AddIn Displayed Name").Installed = True ' or False

End Sub

SectionBreak

Deleting An Add-In

Removing an existing Add-In consists of two steps: (1) Deleting the actual XLA file, and (2) removing the Add-In from the list of add-ins. Close Excel and then delete the actual XLA file from disk. Next, open Excel, go to the *Tools* menu, choose *Add-Ins* and attempt to load deleted add-in. Excel will prompt you with a message *Cannot find add-in: filename.xla. Delete from list?*. Click *Yes* to delete the Add-In from Excel's list of available add-ins.

This page last updated: 6-October-2007