# Building an Excel Add-In

## About Add-Ins

An Excel Add-In is a file (usually with an **.xla** or **.xll** extension) that Excel can load when it starts up. The file contains code (VBA in the case of an **.xla** Add-In) that adds additional functionality to Excel, usually in the form of new functions or macros.

Add-Ins provide an excellent way of increasing the power of Excel and they are the ideal vehicle for distributing your custom functions. Excel is shipped with a variety of Add-Ins ready for you to load and start using (e.g. the *Analysis Toolpak*) and many other third-party Add-Ins are available.

To make use of an Add-In you first have to tell Excel to load it when the program starts. Go to **Tools > Add-Ins** to open the **Add-Ins** dialog (*Fig. 1*). This shows a list of available Add-Ins stored in the default locations. If your Add-Ins are located elsewhere (such as on a network location or in your *My Documents* folder) you can search for them by clicking the dialog's **Browse** button.
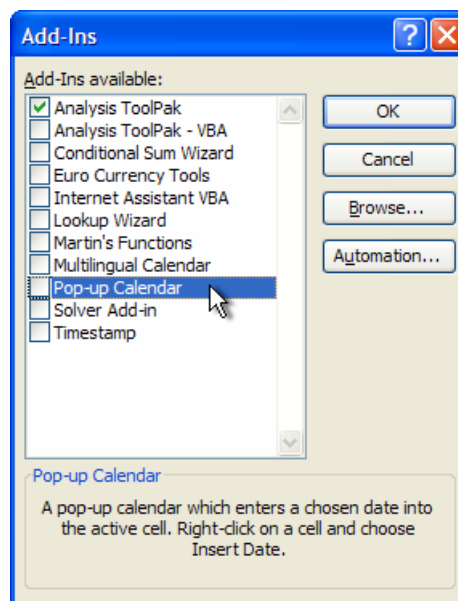


*Fig. 1 Visit the Add-Ins dialog to view and load available Add-Ins.*

Once selected, an Add-In loads each time Excel is started. If you no longer require the Add-In visit the Add-Ins dialog again and deselect it. It will be unloaded when you click the **OK** button.

## Creating an Add-In

An Excel Add-In can contain both custom functions (UDFs) and macros. These are created in an Excel workbook the usual way. You might wish to add some additional features such as toolbar or menu items that are created when the finished Add-In loads and removed then it unloads. When the workbook is ready it is saved as an Add-In. This file can then be distributed and loaded so that people can benefit from the increased functionality brought by your functions and macros.

### Preparing the Source Workbook

Create your macros or custom functions in one or more modules in the usual way. Don't bother adding anything to the workbook's worksheets because in Add-In form these will not ever be seen. If your work takes a while to complete you can save it as a regular Excel workbook file (*.xls) and convert it to an Add-In when you finish.

Write and test your macro or function code. Then add descriptions to each of your custom functions. To do this, in Excel go to **Tools > Macro > Macros** to open the macro dialog box. In the **Macro Name** textbox enter the name of a custom function. Excel will recognise the name of the function and the **Options** button will become enabled (if this does not happen you might have misspelled the name). Click the **Options** button to open the *Macro Options* dialog and enter a brief description of the function. Click **OK** to close the *Macro Options* dialog. You will see your

description now appears in the *Description* area of the *Macro* dialog box (*Fig. 2*). Click **Cancel** to complete the process.
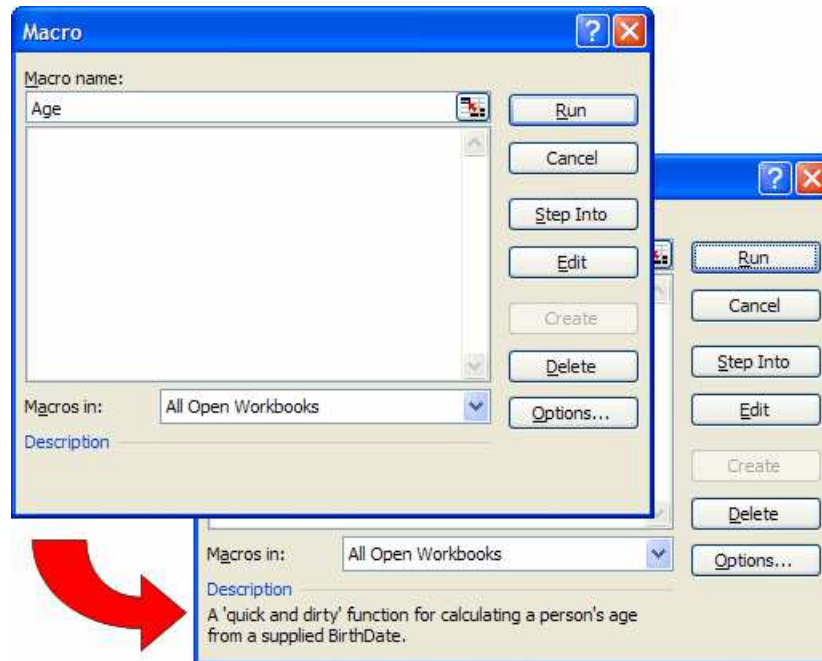


*Fig. 2 Use the Macro dialog to add a description to each custom function.*

Users will be able to access the custom functions in the usual way using Excel's *Insert Function* tool where they will find the functions in the *User Defined* section. But if you have included any macros in your Add-In it is a good idea to provide additional toolbar or menu items to allow them to easily run them.

### Adding Toolbar and Menu Items for Add-In Macros

Suppose you have included a simple macro to insert the current time into the active cell. The macro code looks like this…

```
Sub InsertTime()
    ActiveCell.Value = Format(Now, "hh:nn:ss")
End Sub
```

You can also create an item for the right-click (context) menu of the worksheet so that when the user right-clicks on a cell they can run the macro by choosing the menu item. This requires two code procedures to be written, one to add the menu item to the worksheet context menu when the Add-In is loaded and another to remove it when the Add-In unloads.

The code to create the new menu item should be placed in the *Workbook_Open* procedure which is located in the *ThisWorkbook* code module. The code should look something like this…

```
Private Sub Workbook_Open()
    Dim NewControl As CommandBarControl
' Assign shortcut to insert time on SHIFT+CTRL+T
    Application.OnKey "+^{T}", "basMacros.InsertTime"
' Remove existing shortcut if present
    On Error Resume Next
    Application.CommandBars("Cell").Controls("Insert Time").Delete
    On Error GoTo 0
' Add item to shortcut menu on open
    Set NewControl = Application.CommandBars("Cell").Controls.Add
    With NewControl
        .Caption = "Insert Time"
        .OnAction = "basMacros.InsertTime"
        .BeginGroup = False
    End With
End Sub
```

The above procedure includes a routine to remove the menu item if it is already present (if, for example, Excel did not close properly last time). It adds the text "*Insert Time*" to the existing context menu of the worksheet and also sets the keyboard shortcut **[Shift]+Control]+T** to activate the macro. Note that the macro is referred to by its full address including the name of the module in which it resides (*basMacros.InsertTime*). This is to avoid a conflict if there is already another macro with the same name on the user's computer. The user can now easily run the macro by right-clicking on any cell (*Fig. 3*).
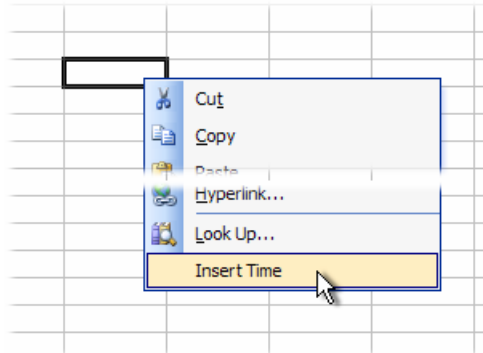


*Fig. 3 A new item has been added to the cell's context menu.*

A second procedure is needed to remove the menu item and cancel the keyboard shortcut when the Add-In unloads. This should be placed in the *Workbook_BeforeClose* procedure and should take the form…

```vba
Private Sub Workbook_BeforeClose(Cancel As Boolean)
' Cancel keyboard shortcut assignment
    Application.OnKey "+^{T}", ""
' Remove shortcut as file closes
    On Error Resume Next
    Application.CommandBars("Cell").Controls("Insert Time").Delete
End Sub
```

Alternatively you might prefer to create a new item on Excel's Menu Bar.

```vba
Private Sub Workbook_Open()
    On Error Resume Next
    Dim MenuBar As CommandBar
    Dim NewMenu As CommandBarPopup
    Dim NewMenuItem As CommandBarButton
' Create new item on Worksheet Menu Bar
    Set MenuBar = Application.CommandBars("Worksheet Menu Bar")
    Set NewMenu = MenuBar.Controls.Add(Type:=msoControlPopup)
    NewMenu.Caption = "Martin's Macros"
' Add items to new menu
    Set NewMenuItem = NewMenu.Controls.Add(Type:=msoControlButton)
    With NewMenuItem
        .Caption = "Insert Time"
        .FaceId = 33
        .OnAction = "basMacros.InsertTime"
    End With
    Set NewMenuItem = NewMenu.Controls.Add(Type:=msoControlButton)
    With NewMenuItem
        .Caption = "Delete Empty Rows"
        .FaceId = 634
        .OnAction = "basMacros.DeleteEmptyRows"
    End With
End Sub
```

The procedure first creates a new menu, in this example it bears the name "*Martin's Macros*", then adds menu items to it. You could use the same technique to add items to an existing menu. In later versions of Excel menu items can also display an icon. There are many icons built in to Excel and each is referred to by a number which is specified as the menu item's *FaceId* property. If you don't want to display an icon next to the menu item simply omit the code line.

You can download a free Add-In which displays a list of available icons and their FaceId numbers from http://skp.mvps.org/faceid.htm

You will also need to remove the menu and its contents when the Add-In unloads. The following code in the *Workbook_BeforeClose* procedure will do the trick. Note that it is not necessary to remove the individual menu items. Removing the menu automatically does that.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
' Delete new menu as workbook closes
    On Error Resume Next
    Application.CommandBars("Worksheet Menu Bar").Controls("Martin's Macros").Delete
End Sub
```

In addition to context menu items and menu bar items, VBA can be used to create new toolbars and command buttons. Here is some typical code…

```
Private Sub Workbook_Open()
    On Error Resume Next
    Dim NewToolbar As CommandBar
    Dim NewButton As CommandBarButton
' Create new toolbar
    Set NewToolbar = Application.CommandBars.Add("Martin's Toolbar")
    NewToolbar.Visible = True
' Add items to toolbar
    Set NewButton = NewToolbar.Controls.Add(Type:=msoControlButton)
    With NewButton
        .FaceId = 33
        .TooltipText = "Insert Time"
        .OnAction = "basMacros.InsertTime"
    End With
    Set NewButton = NewToolbar.Controls.Add(Type:=msoControlButton)
    With NewButton
        .FaceId = 634
        .TooltipText = "Delete Empty Rows"
        .OnAction = "basMacros.DeleteEmptyRows"
    End With
End Sub
```

As before it is necessary to include a procedure to remove the toolbar as the Add-In closes…

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    On Error Resume Next
' Delete new toolbar as workbook closes
    Application.CommandBars("Martin's Toolbar").Delete
End Sub
```

If you create a combination of context menu items, new menus and toolbars you can combine the code to create and remove them into the *Workbook_Open* and *Workbook_BeforeClose* event procedures.

## *Protecting Your Code*

It is a good idea to protect your Add-In code from prying eyes and interfering fingers! First check and double-check everything and test your code rigorously before distributing it. When you are happy that everything works as it should you can add password protection.

In the Visual Basic Editor go to **Tools > VBAProject Properties** then click on the **Protection** tab. Place a tick in the **Lock Project for Viewing** checkbox then enter and confirm a password. Finally click **OK**. The password protection comes into effect when the file is next opened after being saved. Once the file has been saved, any attempt to view the code will be trigger a password request (*Fig. 4*).
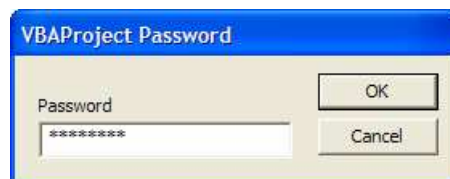


*Fig. 4 The Visual Basic Editor asks for a password.*

WARNING!: If you forget your password you won't be able to get at your code. Consider saving an unprotected copy of the file in a safe place for backup purposes in case you lose the password.

*Saving the Workbook as an Add-In*

The workbook containing your code module now has to be saved as an Excel Add-In (*.xla) file. In the Excel window go to **File > Save** to open the **Save As** dialog. Enter a name for your Add-In file (the usual file naming rules apply) and use the **Save as type** option to change the file type to **Microsoft Excel Add-In (*.xla)** (*Fig. 5*).
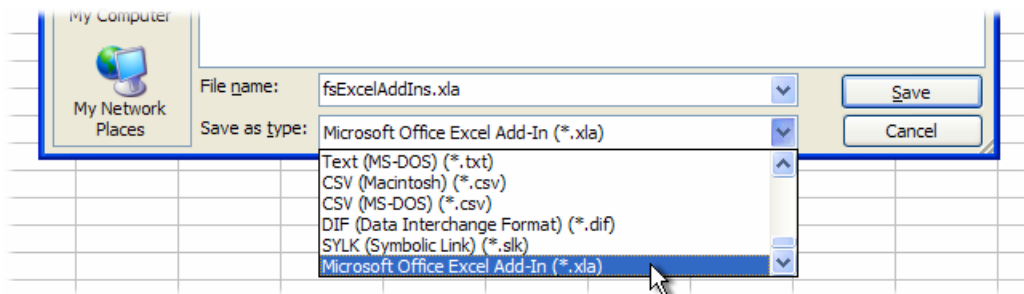


*Fig. 5 Save the workbook as an Excel Add-In.*

Before clicking **OK** check the location in which you are saving the Add-In file. You can store your Add-In anywhere you like but, if you want it to be listed along with the built-in ones, you should save it into the correct location. Excel versions 2000 and later will automatically take you to the correct folder but Excel 97 does not.

Having saved the Add-In you will find that the original workbook you created is still open. You can close this file and discard it if you wish.

*Adding a Description to the Add-In*

It is a good idea to add a description to the Add-In itself. This description will be displayed in the Add-Ins dialog box when you choose an Add-In to install.

First, use the file manager to locate your Add-In file. Right-click on the file icon and choose **Properties** from the context menu. In the file properties dialog click the **Summary** tab. Type a description of your Add-In in the **Comments** text box. If you wish you can also type a name for your Add-In in the **Title** text box. This is useful if you have chosen a short or cryptic name for your *.xla file but would like to show a more descriptive name in the Add-Ins dialog. I could give my Add-In file the filename *fsExcelAddIns.xla* but assign it the title *Martin's Macros & Functions* (*Fig. 6*).
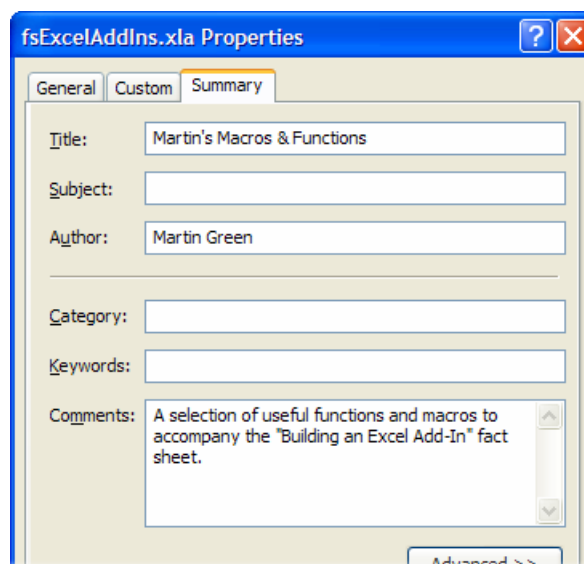


*Fig. 6 Add a Title and description for your Add-In.*

Finally click **OK** to accept your changes. You Add-In is now ready for installation, and can be distributed to other users if required.

## Installing the Add-In

If Excel has not been shut down since you created your Add-In (or since one was copied to the computer's hard disk) restart Excel to make sure that it refreshes its list of available Add-Ins.

Go to **Tools > Add-Ins** to open the **Add-Ins** dialog. If you have stored your Add-In in the default location you will see its name displayed in the **Add-Ins available** window (if you have stored your Add-In in a different folder, use the **Browse** button to find it). Click on your Add-In's name to see its description at the bottom of the dialog box (*Fig. 7*).
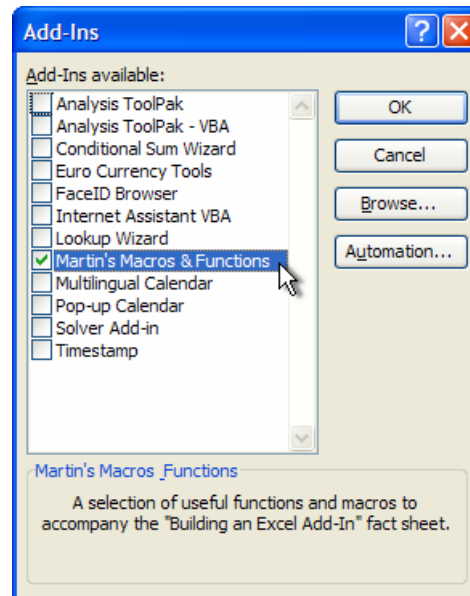


*Fig. 7 Installing the Add-In.*

To install your Add-In place a tick in the check-box next to your Add-In's name and click **OK**. As soon as the Add-In is installed its functions will be available in Excel. Find them in the **User Defined** section of the *Insert Function* tool or simply type them into a cell as you would any built-in function. The Add-In will remain installed until you return to the **Add-Ins** dialog and uninstall it by removing the tick from the check-box.

## Making Additions and Changes to an Add-In

Your Add-In file can contain as many modules and custom functions as you want. You can add them at any time. If your Add-In is installed you will see it listed in the **Project Explorer** pane of the VB editor. Locate the module containing your functions and make whatever additions and changes you want. If your Add-In is not installed, find the Add-In file and double-click it to open it in Excel. You will not be able to see it in the Excel window but it will appear in the VB editor's Project Explorer. Remember to save your changes! Do this from the VB editor window with **File > Save**.

## A Final Word of Caution

A custom function that is located in a code module within a workbook will go wherever the workbook goes. In other words if you open the workbook file on a different machine, or e-mail it to someone else, the function travels with the workbook and will always be available.

If your workbook refers to a custom function contained in an Add-In, the workbook will only be able to calculate the function when the Add-In is present. If you mail the workbook to someone else you will have to mail them the Add-In too.