

## Build an Excel Add-In...

- [VBA Index](#)
- [Printer Friendly version](#)
- Related Topics:
  - [Write Your First VBA Function in Excel.](#)
  - [More Custom Functions.](#)
  - [Using Add-In Functions in Your VBA Code.](#)

## Which Version of Excel?

Most of the screenshots here are from Excel XP (2002), but I have tested this method with Excel 2000 and Excel 97. If there are any significant differences between versions I have provided additional notes and/or screenshots. Using Excel 5 or 95?... Sorry guys, it's time to upgrade!

# Build 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.

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, and many third-party Add-Ins are available.

This article shows you how to write a custom function using Excel VBA and how to save and install it as an Add-In. Custom functions are often referred to as UDFs (User Defined Functions). If you have never built a UDF before, this is a good place to start, or you might like to take a look at the tutorial [Writing Your First VBA Function in Excel](#) which demonstrates the process in detail with some more examples.

In the Excel tutorial [Working Out a Person's Age - An Introduction to Nested IF Statements](#) I showed how to use IF statements to calculate someone's age from their date of birth. Surprisingly, Excel does not have a built-in function for this commonly-required calculation so it is an ideal candidate for a custom function.

If you are already comfortable with writing custom functions, you can go straight to the section explaining how to save your UDFs as an Add-In. [[Jump to Add-In section](#)]

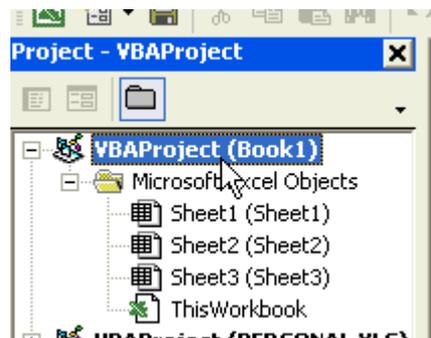
## Write the Function

An Add-In can contain as many UDFs as you want, and you can add more later simply by opening and editing the Add-In file.

### Step 1: Add a Code Module to a New Workbook

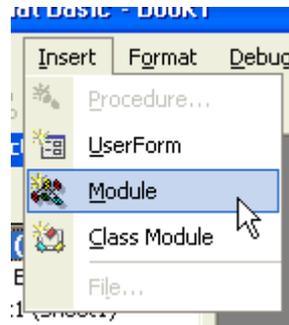
Start Excel or, if you already have Excel open, create a new empty workbook.

Open the Visual Basic Editor from **Tools > Macro > Visual Basic Editor** (Keys: ALT+F11). In the *Project Explorer* pane select **VBAProject (Book1)**. This selects the empty workbook.



If you were already working in Excel the name might be different - check the name in the title bar of the Excel window if you aren't sure. If the Project Explorer is not visible open it by going to **View > Project Explorer**.

From the Insert menu choose Module. This adds a new empty code module to the selected workbook. You will also see the module appear in the Project Explorer pane.



## Step 2: Enter the Code

In the code window type the line...

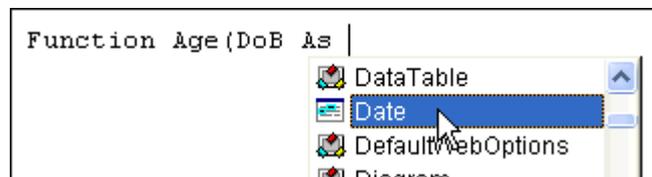
```
Function Age(DoB As Date)
```

...and press ENTER. The VB editor automatically enters the line "End Function" and places your cursor in the empty space between them.

Typing "Function" defines the code that follows as a function, as opposed to a macro or subroutine, which is denoted by the word "Sub".

Next is the function's name, which can be anything you like. Keep your function names simple and descriptive. Spaces aren't allowed so separate words with underscores (e.g. Age\_in\_years) or, preferably, with capitals (e.g. AgeInYears).

A function usually needs one or more "arguments". These are entered in brackets after the function name. An argument is a piece of information that the function uses to perform its calculation. When declaring the arguments it is good practice to specify the data type. In this case only one argument is needed, the date of birth that will be used to calculate the person's age. For simplicity, I have called this argument "DoB". Arguments work like variables in macros. When you type the word "As" after the argument's name the VB editor presents you with a list of possible entries. You can type or pick from the list.



Between the Function and End Function statements, press TAB (to indent your code for clarity) and type the line...

```
    Age = Int((Date - DoB) / 365.25)
```

This tells Excel how to work out the function. It uses two built-in functions, INT (which makes a whole number - or integer - out of the result) and DATE (a visual basic function equivalent to Excel's TODAY() function that returns the current date).

The statement effectively reads... "Take today's date and subtract from it the supplied date of birth. Divide the result by 365.25 and show the answer as a whole number by rounding down."

The finished function should look like this:

```
Function Age(DoB As Date)
    Age = Int((Date - DoB) / 365.25)
End Function
```

## Step 3: Test the Function

You can try out the function right away. Switch to Excel and in your empty workbook (the same one that you are using to create the function's code module) enter a date into a cell. In another cell enter your function in the same way as you would use one of Excel's built-in functions, e.g. **=Age(A1)**

	A	B	C	D
1	27/09/1950	51		
2				
3				

A UDF is available to all open workbooks whenever its host workbook (the workbook containing the UDF's code module) is open. However, if you try to use the function in a different workbook you can encounter a problem...

	A	B	C	D
1	27/09/1950	#NAME?		
2				
3				

The other workbook can't find the function so the **#NAME?** error appears. To avoid this happening you could add the host workbook's name to the function to tell Excel where to find it...

	A	B	C	D
1	27/09/1950	51		
2				
3				

This works but is very clumsy and demonstrates one of the reasons why the best place for your custom functions is inside an Excel Add-In (with some exceptions... see [note below.](#))

## Adding Features to the Function

### Allowing for the Absence of Data

A disadvantage of many functions is that when you prepare a worksheet in advance of its receiving data (e.g. in a template where you want the functions to be in place ready to calculate the user's input) the functions try to calculate on empty cells, often producing errors or nonsense results. If our **Age** function tries to calculate an empty cell it still produces a result, in this case **102** (the current year being 2002). This happens because it takes the value of the empty cell to be zero, and interprets day zero as *January 0 1900*. This is logical to Excel because the first day it knows about is day 1, which is January 1 1900.

Normally you would try to anticipate this by entering your age function as part of an IF statement e.g. **=IF(ISBLANK(A1),"",age(A1))** The IF statement tells Excel to enter a value of "nothing" (represented by "") if the data cell is blank but to calculate the **Age** function if it contains a value...

	A	B	C	D	E
1					
2					
3					

	A	B	C	D	E
1	27/09/1950	51			
2					
3					

This works fine, but you can save yourself the trouble by incorporating this sort of troubleshooting into the code of the function itself.

Modify the function code as follows:

```

Function Age(DoB As Date)
    If DoB = 0 Then
        Age = ""
    Else
        Age = Int((Date - DoB) / 365.25)
    End If
End Function

```

Alternatively a custom error message could be included by replacing the pair of quote marks in code line 3 with a message in quotes e.g. **Age = "No Birthdate"**. The result would be...

The screenshot shows an Excel spreadsheet with a formula bar at the top displaying `=age(A1)`. The spreadsheet has columns labeled A, B, C, and D, and rows numbered 1, 2, and 3. Cell B1 is selected and contains the text "No Birthdate".

	A	B	C	D
1		No Birthdate		
2				
3				

## Making the Function More Accurate

The calculation used in the example above is very accurate, but not completely accurate. It works on the principle that there is an average of 365.25 days in a year (usually 365 but 366 every fourth year) so dividing a person's age in days by 365.25 should give their age in years.

This works fine most of the time but it can (rarely) throw up an error. If the person in question has their birthday today and were born on a year that is a multiple of 4 years ago, the calculation will be a year out. A small possibility, but if we're going to do it we might as well do it right!

In my tutorial [Working Out a Person's Age - An Introduction to Nested IF Statements](#) I showed how to use IF statements in Excel to calculate someone's age from their date of birth, with complete accuracy. I could do the same in VBA for my custom function (although the syntax of a VBA IF statement is slightly different to that in Excel) but I prefer to use a CASE statement. Excel doesn't have the CASE statement but VBA does. I find CASE statements easier than IF statements to figure out when the logic is slightly complex.

Here is the code for my improved function:

```

Function Age(DoB As Date)
    If DoB = 0 Then
        Age = "No Birthdate"
    Else
        Select Case Month(Date)
            Case Is < Month(DoB)
                Age = Year(Date) - Year(DoB) - 1
            Case Is = Month(DoB)
                If Day(Date) >= Day(DoB) Then
                    Age = Year(Date) - Year(DoB)
                Else
                    Age = Year(Date) - Year(DoB) - 1
                End If
            Case Is > Month(DoB)
                Age = Year(Date) - Year(DoB)
        End Select
    End If
End Function

```

**TIP:** Select the lines of code in the grey box above, copy them (Keys: CTRL+C) and paste them directly into your VBA code window (Keys: CTRL+V).

## How the code works...

**Function** Age(DoB As Date)

Gives a name to the function and declares that a single argument is needed, which must be a

date.

If DoB = 0 Then  
Age = "No Birthdate"

An IF statement to determine whether there is a value in the data cell. The value of an empty cell is read as zero. If that is true then the function returns the text "No Birthdate".

Else  
Select Case Month(Date)

If the data cell is not empty, consider the month that it is today...

Case Is < Month(DoB)  
Age = Year(Date) - Year(DoB) - 1

If today's month is before (i.e. less than) the month of the person's date of birth, they have not had their birthday, so their age is this year minus their birth year minus 1.

Case Is = Month(DoB)

If today's month is the same as the month of the person's date of birth we need to know whether or not they have had their birthday yet so...

If Day(Date) >= Day(DoB) Then  
Age = Year(Date) - Year(DoB)

If the today is equal to or past the day of their birthday, then they have had their birthday (or it is today) so their age is this year minus their birth year...

Else  
Age = Year(Date) - Year(DoB) - 1  
End If

...otherwise, they have not had their birthday so their age is this year minus their birth year minus 1.

Case Is > Month(DoB)  
Age = Year(Date) - Year(DoB)

If today's month is after (i.e. greater than) the month of the person's date of birth, they have had their birthday, so their age is this year minus their birth year.

End Select  
End If  
End Function

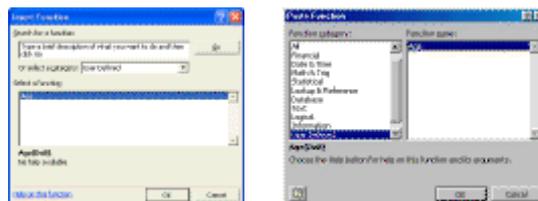
Closes the CASE statement, the IF statement and the Function.

This calculation may seem rather complex but you only have to type it once! When you have created your function all you ever have to type is its name.

## Creating an Excel Add-In

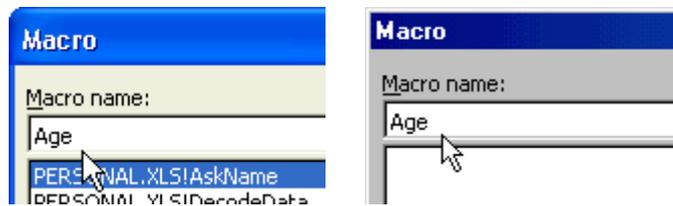
### Step 1: Adding a Description to the Function

When a function is inserted via the function wizard (i.e. the Paste Function tool) the user sees a description of the function that helps them choose the correct one. This isn't the case with custom functions [click the thumbnail below to see a full-sized image]...



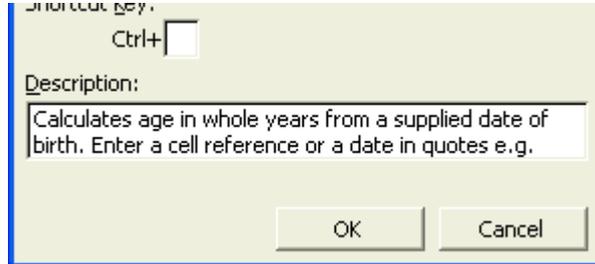
If you want a description you have to add it yourself. Here's how...

Go to **Tools > Macro > Macros** to open the **Macro** dialog box. You will see all available macros listed but no custom functions. In the **Macro name:** text box type the name of the function you want to describe...

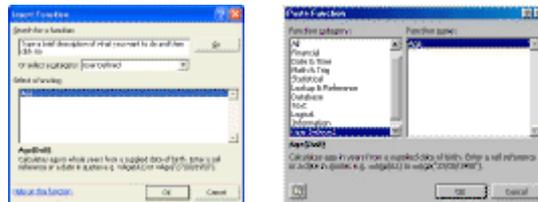


Click the **[Options]** button to open the **Macro Options** dialog box. (If the Options button is greyed out the function name you entered has not been recognised. Check your typing.)

Type a description for your function in the **Description:** text box...



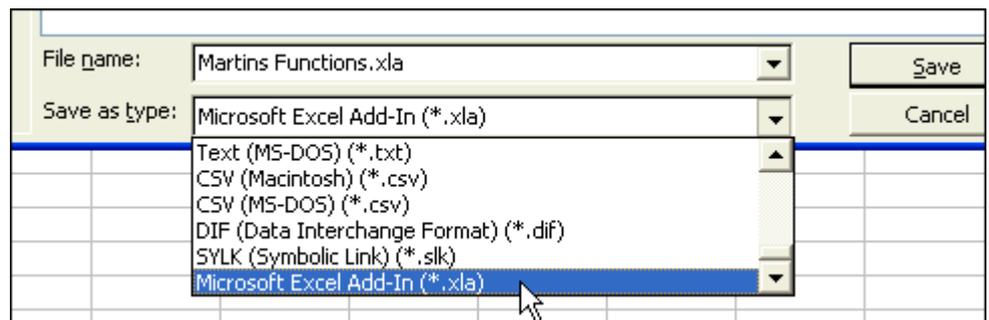
Click **[OK]** to close the Macro Options dialog and then click **[Cancel]** to close the Macro dialog. The description has now been assigned to the function. You can repeat the procedure to amend the description if required. This description is now shown when the custom function is selected in the Function Wizard [click the thumbnail below to see a full-sized image]...



## Step 2: Save 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)**.



Before clicking **[OK]** check the location in which you are saving the Add-In file. This will vary depending upon your particular set-up. On my computers the locations are:

### Excel XP(2002) on Windows XP:

C:\Documents and Settings\UserName\Application Data\Microsoft\AddIns

### Excel 2000 on Windows 98:

C:\Windows\Application Data\Microsoft\AddIns

### Excel 97 on Windows 98:

C:\Program Files\Microsoft Office\Office\Library

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 XP and Excel 2000 will automatically take you to the correct folder but Excel 97 does not.

**TIP:** To check the save location, first set the **Save as type:** to **Microsoft Excel Add-In** then open the **Save in:** drop-down list to reveal the path to the save folder.

You can now close the original workbook (Book 1). You do not need to save the changes if prompted.

### Step 3: Add 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 **mgfunctions.xla** but assign it the title **Martin's Functions**.

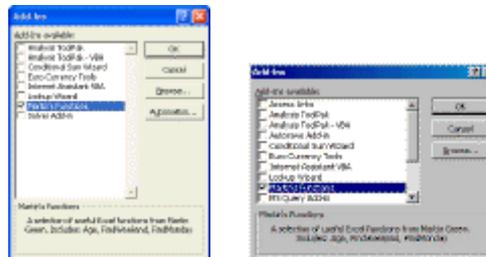
Finally click **[OK]** to accept your changes. Your Add-In is now ready for installation, and can be distributed to other users if required.

### Step 4: Install 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.

To install your Add-In place a tick in the check-box next to your Add-In's name and click **[OK]**. [Click the thumbnail below to see a full-sized image]...



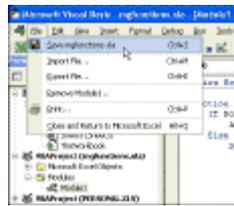
As soon as the Add-In is installed its functions will be available in Excel. Find them in the **User Defined** section of the Function Wizard (Paste Function Tool) or simply type them into a cell as you would any built-in function. The Add-In will remain installed until you return to the **Add-Ins** 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** [click the thumbnail below to see the full sized image]...



## Adding Comments to Your Code

It's easy to add notes (properly called "comments") to your VBA code. Enter an apostrophe (i.e. single quote mark) and then type. The apostrophe tells the code compiler to ignore everything that follows it on the same line so it isn't treated as executable code and the compiler doesn't try to run it. Anything you type after the apostrophe is "commented out" and is coloured green in the VB editor's code window...

```
Function Age(DoB As Date)
' =====
' Calculates age in whole years from a supplied
' Author: Martin Green (martin@fontstuff.com)
' =====
If DoB = 0 Then
    Age = "" 'Returns nothing if no date supplied
Else
    Age = Int((Date - DoB) / 365.25)
```

You can use this technique to add explanatory notes to your code (ever returned to some code you wrote a while ago and wondered what on earth it means?).

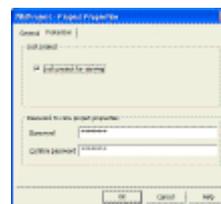
You can also temporarily comment out lines of code so that they don't run - useful for testing your code. If you think you don't need a line of code, don't delete it but comment it out first. Then test your code to see if it still works. If everything is OK you can now delete the line. If your code fails, simply remove the apostrophe and your line of code is restored.

## Password Protecting Your Code

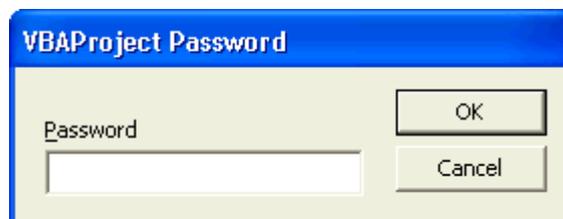
VBA allows you to password-protect your code, whether it is the macro code behind a regular workbook or the code modules of an Add-In. Password-protecting code prevents unauthorised viewing of your code for purposes of security or copyright.

To password-protect your Add-In locate it in the VB Editor. Right-click its name in the Project Explorer pane and choose **VBAProject Properties...** from the context menu. In the **VBAProject - Project Properties** dialog click the **Protection** tab.

Place a tick in the **Lock project for viewing** check box, then enter and confirm your password. Click **[OK]** and go to **File > Save** to save your changes. [Click the thumbnail below to see the full-sized image]...



The password-protection will take effect from the next time Excel is opened. Any attempt to view the Add-In's code module will result in a request for the password...



# Assigning Add-In Macros to Custom Toolbar Buttons and Menu Items

Add-Ins aren't just for containing functions. They may contain useful macros. You may want to assign a macro that is contained in an Add-In to a custom toolbar button or custom menu item.

Normally to do this you would right-click on your new button or menu item and choose **Assign Macro** and pick your macro from the list provided. However, the names of macros contained in Add-Ins are not shown in the list. Don't worry! Just type the name of the macro in the **Macro Name:** box at the top of the dialog. Excel will find it and the macro will run as expected when the button is clicked or the menu item chosen.

## 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!

If you want to use custom functions contained in an Add-In in a the code of another workbook, you will have to set a reference to the Add-In. Read the article [How to Use Your Excel Add-In Functions in VBA](#) for details.

[^ top](#)