In Full Colour

Curtis Frye

brilliant

Microsoft® Excel VBA Programming

what you need to know and how to do it

Brilliant Excel VBA Programming

Curtis Frye

PEARSON

Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Auckland • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • São Paulo • Mexico City • Madrid • Amsterdam • Munich • Paris • Milan

Add a worksheet

Add a worksheet

1. Create a subroutine.
2. In the body of the subroutine, type **`ThisWorkbook.Sheets.Add`**
3. If desired, use any of the following parameters to specify where and what type of sheets to add:
   a. **`Before`**
   b. **`After`**
   c. **`Count`**
   d. **`Type`**

Important

You may use either the **`Before`** or **`After`** argument, but not both.

For your information

You *can* add any number of worksheets to your workbook, but it's best not to add more than you need.

Every Excel workbook must contain at least *one* worksheet, but most workbooks will contain multiple worksheets. For example, if you store a year's worth of data in a single workbook, you should consider creating a worksheet for each month. Doing so divides your data into manageable units and lets you find the specific data you're looking for more easily.

To add a sheet to a workbook, you use the **`Sheet`** collection's **`Add`** method. For example, if you wanted to add a worksheet to the same workbook that contains your VBA code, you could use the code snippet **`ThisWorkbook.Sheets.Add`**. You can also use several other parameters to identify the position of the sheet you add, the number of sheets to be added and type of sheet to be added.

- **`Before`** identifies the existing sheet before which you place the new sheets. If you leave this parameter out, Excel adds the sheet before the active sheet.
- **`After`** identifies the existing sheet after which you place the new sheets.
- **`Count`** indicates the number of sheets to be added.
- **`Type`** identifies the type of sheet to be added to your workbook. You can select from the sheet types **`xlWorksheet`**, **`xlChart`**, **`xlExcel4MacroSheet`** and **`xlExcel4IntlMacroSheet`**. If you leave this parameter blank, the **`Add`** method inserts a worksheet.

As an example, you could use the following code to add two worksheets after the sheet named Sheet2:

```
ThisWorkbook.Sheets.Add
After:=Worksheets("Sheet2"), Count:=2
```

This variation of the code would add a chart sheet at the beginning of the workbook:

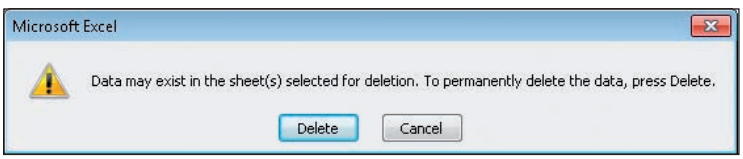
```
ThisWorkbook.Sheets.Add
Before:=Worksheets(1), Type:=xlChart
```

Excel workbooks are similar to many other projects in the sense that you are never truly done changing them. Whether you add new data, change the formulas on a worksheet or modify worksheet formatting, you will probably find new ways to work more effectively. If you find that your changes make one of your worksheets redundant, you can delete that sheet.

The VBA command to delete a worksheet relies on the **`Sheets`** collection's **`Delete`** method. All you need to do is identify the sheet you want to get rid of, either by the number of the sheet within the workbook or by entering the sheet's name. The following two code snippets provide an example of each approach:

```
Sheets(1).Delete
Sheets("Sheet1").Delete
```

When you attempt to delete a worksheet that contains data, whether by using the user interface or VBA, Excel displays a confirmation dialog box asking if you're sure you want to delete the worksheet. You can temporarily disable alert boxes by adding the command **`Application.DisplayAlerts = False`** on a line before you invoke the Delete method.



Delete a worksheet

Delete a worksheet

1. Create a subroutine.
2. In the body of the subroutine, use one of the following code patterns:
   a. **`Sheets(1).Delete`**
   b. **`Sheets ("sheetname"). Delete`**

For your information

Be sure to set the **`DisplayAlerts`** property to **`True`** after you delete the worksheet. Not doing so could cause you to miss other important warnings.

Did you know?

You can also use an InputBox (see Chapter 11) to enter the name of the worksheet to be deleted.

Move a worksheet

Move a worksheet

1. Create a subroutine.
2. In the body of the subroutine, use one of the following code patterns:
   a. `Sheets(1).Move` – which moves the numbered worksheet to a *new* workbook.
   b. `Sheets(2).Move, Before:=Sheets(1)` – moves the numbered worksheet *before* the first worksheet.
   c. `Sheets(1).Move, After:=Sheets(3)` – moves the numbered worksheet *after* the third worksheet.

You will often find that the data contained in one workbook could be useful in another. If that's the case, you can move a worksheet to another workbook or, if you find your workflow isn't as efficient as it might be, you can relocate a worksheet within the same workbook. Moving a worksheet doesn't leave a copy of the worksheet in its original position – as the name implies, it cuts the worksheet from its original position and pastes it in its new position.

You can move a worksheet quickly using Excel VBA by using the **`Sheets`** collection's **`Move`** method. Using the **`Move`** method by itself, without indicating a destination for the sheet you're moving, causes Excel VBA to move the sheet to a new workbook. If you want to move the worksheet within the current workbook, you can use one of the **`Move`** method's two optional parameters: **`Before`** and **`After`**.

- **`Before`** identifies the existing sheet before which you place the moved sheets. If you leave this parameter out, Excel moves the sheet to before the active sheet.
- **`After`** identifies the existing sheet after which you place the moved sheets.

For your information

If you try to move a worksheet after a worksheet that doesn't exist – such as Sheet(4) in a workbook with three worksheets – the **`Move`** method will generate an error.

Did you know?

You can also use worksheet names, enclosed in double quotes, instead of sheet numbers in these commands. For example, **`Sheets("January").Move`**.

Just as you can move a worksheet to another workbook or within the same workbook, you can create a *copy* of a worksheet and move it. For example, you could use a worksheet as a template and copy it within your existing workbook. You can also use copying to include a data set in another workbook without deleting the original worksheet.

If you use the **`Copy`** method by itself without indicating a target destination for the sheet you're copying, Excel VBA copies the sheet to a new workbook. If you want to copy the worksheet within the current workbook, you can use one of the **`Copy`** method's two optional parameters – **`Before`** and **`After`**.

- **`Before`** identifies the existing sheet before which you place the copied sheet. If you leave this parameter out, Excel copies the sheet and places it before the active sheet.
- **`After`** identifies the existing sheet, after which Excel will place the copied sheet.

Did you know?

As with moving worksheets, you can also use worksheet names, enclosed in double quotes, instead of sheet numbers in these commands. For example, **`Sheets("January").Copy`**.

For your information

Trying to copy a worksheet to a position before or after a worksheet that doesn't exist will result in an error. Be sure you create error-handling code to manage these situations (see Chapter 13).

Copy a worksheet

Copy a worksheet

1. Create a subroutine.
2. In the body of the subroutine, use one of the following code patterns:
   a. **`Sheets(1).Copy`** – copies the numbered worksheet to a new workbook.
   b. **`Sheets(2).Copy, Before:=Sheets(1)`** – copies the numbered worksheet to before the first worksheet.
   c. **`Sheets(1).Copy, After:=Sheets(3)`** – copies the numbered worksheet to after the third worksheet.

Hide or unhide a worksheet

Even if you work in a home-based business, your workbooks might contain data that you don't want to display to anyone who might use your computer. You can keep your data away from casual observers by hiding a worksheet. *Hiding* a worksheet doesn't delete it, so you can still use its contents in your formulas, but it does make if a little bit more difficult to find the data unless you know what you're looking for.

The **`Visible`** property indicates whether a worksheet appears in the body of the workbook or not. You can both read the **`Visible`** property to discover if a worksheet is visible or hidden and change the property's value to control whether or not the worksheet appears on the tab bar. The following two code snippets are valid uses of the **`Visible`** property. The first hides Sheets(1) and the second displays the sheet named Sheet3.

```
Sheets(1).Visible = False
Sheets("Sheet3").Visible = True
```

Hide or unhide a worksheet

1. Create a subroutine.
2. In the body of the subroutine, type one of the following lines of code. The first example hides the worksheet, the second unhides it:
   a. **`Sheets(1).Visible = False`**
   b. **`Sheets(1).Vislble = True`**

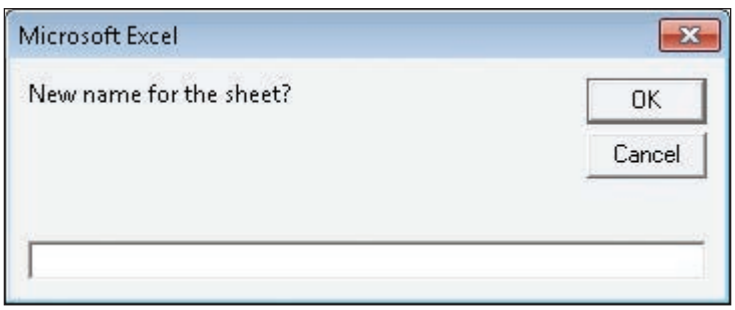
Did you know?

Users can still unhide a worksheet by clicking the View tab, then unhide on the ribbon and selecting a hidden sheet from the dialog box that appears.

When you create an Excel 2010 workbook, it contains three sheets named Sheet1, Sheet2 and Sheet3. These names aren't very descriptive, so the program lets you rename your worksheets. If you rename a worksheet as part of an automated process, you might use the month the data represents, the name of a product or the name of a customer. Doing so makes it easier to recognise each worksheet's contents when you look through the workbook.

You can encode the new name for a worksheet in your VBA routine, but it's more likely that you'll want the flexibility to name the new worksheet by typing in a value. To allow you and your colleagues to do that, display an InputBox and use the control's output for the sheet's new name. One such code snippet might be:

```
Dim strName
strName = InputBox("New name for the
sheet?")
ActiveSheet.Name = strName
```



Important

You cannot have two worksheets with the same name within a workbook.

Rename a worksheet

Rename a worksheet

1. Create a subroutine.
2. In the body of the subroutine, type the following code:

   a. `Dim strName`

   b. `strName = InputBox("New name for the sheet?")`

   c. `ActiveSheet.Name = strName`

See also

For more information on using InputBoxes, see Chapter 11.

5