How to…
Write applications using Visual Basic

Last month, we enhanced our Units application, making it more robust by preventing the user from performing any actions that would cause it to crash. Now it's time to add some extra features such as being able to convert between various types of units.

Open the project as you left it last month. If you didn't save your changes from last month or you have only just started reading this tutorial, you can find a copy of the project files on this months's CD in (ED: Please insert the path to the project files here)

Assuming you have loaded the project, resize the form and reposition the controls as shown in the screenshot opposite. Then, rename the controls on the form as indicated in the following table:

Old Name	New Name
Label1	lblFromUnits
Label2	lblToUnits
txtInches	txtFromUnits
txtCM	txtToUnits

We didn't give Label1 and Label2 "proper" names when we originally created them since their purpose in life was just to display some text that never changed. However, we'll alter the code so that the text the labels display changes depending on what type of conversion is chosen. Therefore, we have now given them "proper" names to help us remember what they refer to ("from" units and "to" units). Set the *Caption* properties of the two labels, *lblFromUnits* and *lblToUnits* to *<From>* and *<To>* respectively. These artificial captions will be replaced by our code later on.

Decisions, decisions

Now, we need a way to let the user decide which type of conversion is going to take place, for example from miles to kilometres. A combo box is perfect for this task. Combo boxes are drop-down lists from which the user can choose an option. Place a combo box in the right-hand region of the blank space you've created on the form, above the button and text boxes. Change the combo box's name to *cboConversionTypes* and its *Style* property to *2 - Dropdown List*. This will stop the user from typing his/her own choices into the combo box since we only want the user to choose from the options that we specify. Place a label control to the left-hand side of the combo box and set its *Caption* property to *Conversion Type*. The form should now look something like the screenshot opposite.

Disappearing Code

Now, double-click on txtFromUnits – notice that the code that used to be there is now gone! Where is the code you typed last month to reject the invalid keypresses? In fact, it is still there, but the event handler is still named after the old name for this control (txtInches). Switch to full-module view if you're not already there and you will see the old code under the event handlers *txtInches_KeyPress* and *txtInches_Change*. Whenever you rename any controls, VB won't automatically change any code to suit the new name(s) – that is left up to you. Delete the skeleton code for *txtFromUnits_Change* that VB has created. Next, change the event handler procedure names from *txtInches_KeyPress* and *txtInches_Change* to *txtFromUnits_KeyPress* and *txtFromUnits_Change* respectively. If you close the code window and then double-click the txtFromUnits textbox, you'll see that VB has taken note of your changes.

The observant amongst you might have guessed that the code in the event handler txtFromUnits_Change won't work either since it will be trying to access txtFromUnits via its old name. Change the reference to txtInches in txtFromUnits_Change to *txtFromUnits* so that the code will work with the new name for the control.

Gotta be Explicit

We need to use some variables to keep track of the various conversion types that will be available. This is the first time we've dealt with any variables. Like many other versions of BASIC, VB doesn't require you to declare variables before you use them. This can lead to problems if you mis-spell a variable name, since VB will use a default value in place of the mis-spelt variable. Fortunately, there is a cure – it is called "Option Explicit". If you type this in at the top of every code window, it will force VB to check that you've declared all your variables before using them. With this in place, VB will tell you if it finds a reference to an undeclared variable, rather than plodding onwards assuming a default value. Move to the very top left-hand corner of the code window, press enter to insert a blank line and then type `Option Explicit` in the blank space you've just created. You can have VB do this for you automatically every time you create a new form, module, class or control (modules, classes and controls will be covered in future tutorials). To enable this feature (highly recommended), choose *Options…* from VB's *Tools* menu and place a tick in the box next to *Require Variable Declaration*. This will be remembered for any future VB sessions that you will use.

Keeping Track

Now, we need a way to keep track of the following things:

- The *name* of the units that we're converting *from*
- The *name* of the units that we're converting *to*
- The *multiplication factor* that we need to multiply our *from units* by to arrive at the value for *to units*

We can use three arrays to keep track of this information, an array to store the *from units* names, an array to store the *to units* names and an array to hold the *multiplication factors*.

When you declare a variable in a VB program, the *place* where you declare it determines which other procedures can see it. If you declare a variable inside a procedure (procedure-level) such as an event handler, nothing outside of that procedure will be able to see it. Since we want the arrays to be visible to a variety of procedures, we need to declare them in a place where other procedures on the form can see them. This place is called the module level, which sits at the very top of the code window, where you typed *Option Explicit*. Module-level, incidentally, has nothing to do with *modules*, which are pieces of VB code that are global to the whole application (these will be discussed in later tutorials). Type the following under *Option Explicit*:

```
Private mastrFromUnitNames(0 To 3) As String
Private mastrToUnitNames(0 To 3) As String
Private masngMultiplicationFactors(0 To 3) As Single
```

I have declared them as being Private (as opposed to Public) since I don't want any other forms or modules to be able to see them (not that there are any other forms or modules yet). *Private* and *Public* are only used at module level; the *Dim* keyword is used to declare variables inside a procedure. The *0 To* bits aren't strictly necessary but I like to put them in for clarity, to remind me that the arrays run from element 0 and not 1.

Strange Names

The strange *mastr/masng* you can see on the front of the variable names are the prefixes I use for naming variables. Unlike other versions of BASIC, you don't need $ signs after strings or % signs after integers. You can declare them using *Private/Public/Dim <VariableName> As <Type>* instead, which I find much more readable. However, variable types are easily forgotten so I prefix them with *str* and *sng* for *String* and *Single* respectively (a *single* is a type of variable that doesn't need to be a whole number). This is similar to the way that we prefixed the textboxes with *txt* to help us remember that we were dealing with textboxes. The *a* on the front of the *str* helps me remember that I'm referring to an array instead of a normal variable. Finally, the *m* means "module-level" to remind me that the variables have been declared at module level. Some people like these standards, others hate them. It's a matter of taste, but I recommend them because if you're in the middle of a very large VB project and you see a reference to a variable, it will be obvious whereabouts it was declared and what type you declared it as. Finally, we'll need a way to remember *which* conversion the user chose from our combo box. Place the following line immediately under the array declarations:

```
Private msngChosenFactor As Single
```

This variable will hold the multiplication factor that corresponds to the conversion choice the user made.

Populating the arrays

Now the variables are declared, we need a place where we can populate (fill) the arrays and combo box with the appropriate values. This needs to be done before our application appears because we don't want the poor user staring at an empty combo box. VB provides us with an event that is perfect for this – the *Form_Load* event. This event occurs when a form is read into memory for the first time, such as when you start your application. This event occurs before the form is shown, however. Close the code window and then double-click on a blank area of the Form. VB correctly assumed that you wanted to place some code into the Form_Load event and has created the necessary skeleton code for you. Type the following into the Form_Load event handler:

```
Dim intIndex As Integer

masngMultiplicationFactors(0) = 1.6093
mastrFromUnitNames(0) = "Miles"
mastrToUnitNames(0) = "Kilometres"

masngMultiplicationFactors(1) = 0.9144
mastrFromUnitNames(1) = "Yards"
mastrToUnitNames(1) = "Metres"

masngMultiplicationFactors(2) = 0.3048
mastrFromUnitNames(2) = "Feet"
mastrToUnitNames(2) = "Metres"

masngMultiplicationFactors(3) = 2.52
mastrFromUnitNames(3) = "Inches"
mastrToUnitNames(3) = "Centimetres"

For intIndex = 0 To 3
    cboConversionTypes.AddItem mastrFromUnitNames(intIndex)
& " to " & mastrToUnitNames(intIndex)
Next

'Pre-select list item number 0
cboConversionTypes.ListIndex = 0
```

Now, run the program by clicking the "Play" button on VB's toolbar. Try clicking on the combo-box to show its drop-down list of available conversions. Nothing will actually work yet, since the code for using the new conversion methods has not been written. I have left a deliberate mistake in the program – see if you can find it (Hint: Think about the controls we renamed earlier).

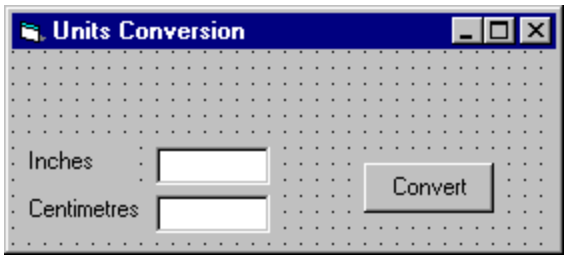
Next month, I'll show you how we can get VB to find the mistake for us automatically. In addition, I'll explain how the code in Form_Load works and we'll add the necessary pieces of code to get the conversions working.

Until then, enjoy the bug-hunt

Nick.

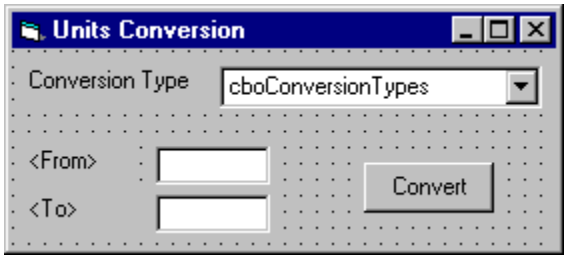
Nicholas Scott is a freelance columnist who currently works for MIS Computer Services in Northwich. Nick can be contacted via email at nicks@miscs.com.

(ED: The filename for this image is “Image1_ResizedFormAndControls.bmp”. Please placed this close to the top of the tutorial near the third paragraph)



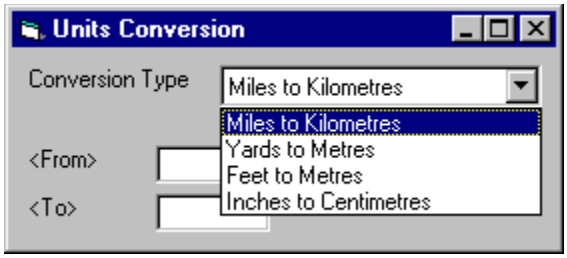
The resized form with its re-arranged controls, making way for the new additions.

(ED: The filename for this image is “Image2_AddedComboAndLabel.bmp”. Please place this near the end of the text under the section “Decisions, decisions”)



The improved version of our form. Hopefully, your version should look similar to this. The user will be able to choose a type of conversion from the combo box we’ve just added.

(ED: The filename for this image is “Image3_ComboList.bmp”. Place this as you see fit, somewhere towards the end of the tutorial.)



Here is the partly-completed form in all its glory. However, you’ll need to wait until next month when we’ll add some code to make the conversion types have any affect.

(ED: Don’t forget to insert the appropriate path to the project files on the cover CD, mentioned at the top of this document)