

## How to...

# Write applications using Visual Basic

Last month, we enhanced our Units application by adding a combo box, to allow the user to choose which types of units to convert between. However, we didn't add any code to take advantage of this new feature. This month, we'll add the remaining code and I'll explain the new terms used in the `Form_Load` event handler you entered last month.

Open last month's project (you did save it didn't you?). As usual, you can find a copy on this cover CD in (ED: Please insert the path to the project files here).

### **Filling in the blanks**

Double-click on an empty part of the form to display its `Form_Load` event handler. You will see the code that we entered last month to populate (fill) the arrays with information about the various types of conversions that can take place. Notice that *Dim* was used to declare the *intIndex* variable since we are declaring it inside a procedure. Had this variable been declared at module level, *Public* or *Private* would have been used instead.

Next, we place the conversion unit names and multiplication factors into the arrays. Readers who have used older versions of BASIC may remember the *Read* and *Data* statements, which would have provided a more elegant way of placing the values into the arrays. Unfortunately, VB does not provide these facilities so we have to do it the hard way. Then, the code loops through the arrays and adds their contents to the combo box using the *AddItem* method.

### **Methods vs Properties**

Methods are similar to *properties*, except that methods normally perform some action on a control, rather than altering its behaviour or obtaining information as a property would.

For example, consider an imaginary "car" control that would emulate a car. This car control would have one method (*StartEngine*) and one property (*FuelRemaining*). *StartEngine* would be a *method* because it performs an action with the car, in this case, starts the engine. *FuelRemaining* on the other hand, would be implemented as a *property* since it merely conveys information (the amount of fuel remaining).

Notice the user of the ampersand (&) operator to *concatenate* (join together) the contents of the two arrays with the text " to ". Previous versions of BASIC used the "+" operator to concatenate strings, but VB uses the ampersand to achieve the same thing.

### **Fooling VB**

Finally, the code pre-selects the first item in the combo box by setting its *ListIndex* property to 0. Items in a combo box are referred to by their indices, 0 being the first, 1

being the second, and so on. Setting this property will cause a click event to be raised for the combo box, to fool VB in to thinking that the user has made a choice. VB will in turn, run the click event handler on the combo box, which we haven't written yet. As you will have gathered, if there is no code provided for an event handler, nothing will take place. We'll add the necessary code in a moment.

Finally, notice that we used an apostrophe (') to add a comment to the code, explaining what the line did. Older version of BASIC used the *Rem* statement to achieve the same thing. You can use *Rem* if you want, but I prefer the apostrophe since it looks less cluttered.

## Watch those events

Now it's time to add the code for the *Click* event handler for our combo box. Close the code window to return to the form. Double-click the combo box to view its event handler. VB has incorrectly assumed that we wanted the *Change* event and has created some skeleton code for us. Choose *Click* from the right-hand combo box in the code window, and then enter the following code

```
Dim intIndex As Integer
intIndex = cboConversionTypes.ListIndex

lblFromUnits = mastrFromUnitNames(intIndex)
txtFromUnits = ""
lblToUnits = mastrToUnitNames(intIndex)
txtToUnits = ""

msngChosenFactor = masngMultiplicationFactors(intIndex)
```

This code determines which item was chosen from the combo box by examining the combo box's *ListIndex* property. Reading *ListIndex* merely gets its value, unlike setting *ListIndex*, which would cause the *Click* event to run, like we did in *Form\_Load*. The index we obtained is used to look up the corresponding "from" and "to" unit names, and the appropriate text is placed into the *Caption* properties of the two label controls. Next, the *Text* properties of the two textboxes are emptied to clear any old entries. Finally, the corresponding multiplication factor is stored for later use in *msngChosenFactor*.

## Default Properties

Notice that I didn't explicitly state that I wanted to use the *Caption* properties of the labels, nor did I mention the *Text* property of the textboxes. That is because the *Caption* property is the *default* property of a label, and the *Text* property is the default property of a textbox. Since they are defaults, VB assumes that they are the properties I wanted, and uses them as appropriate. In fact, the code in the *Click* event handler of the *cmdConvert* button didn't need the *Text* properties of the two text boxes to be explicitly mentioned either. I included these properties when we started off because it made the code obvious

as to which properties it was referring. Now we're getting more proficient at VB, I'll show you other shortcuts like these. There are some circumstances where you would *have* to include the Text property even though it is the default – I'll point these out and explain them when we encounter them later in this series.

Now, remove the skeleton code for the *Change* event that VB mistakenly created for us earlier.

## Bugs that Bite

Run the program now by clicking the “Start” button on VB's toolbar. Notice that the first item in the combo box has already been selected for us, courtesy of the final line in `Form_Load`. You can also see that the label captions *<From>* and *<To>* have been replaced with the names of the respective units, thanks to the code on the combo box's click event handler. Enter a value into the topmost textbox (the “from units” field) and then click on the Convert button. The application crashes with an error – “Variable not defined”. This is because the code in the click event handler of `cmdConvert` is still using the old control names. This is the deliberate mistake I left you to find last month. Dismiss VB's dialog box by clicking *Ok*, and then click on the “End” button on VB's toolbar. The point of this is that you didn't find out about this problem until VB tried to run the code that contained the old names.

There is a way to get VB to perform a more thorough check throughout all of your code before actually running the application. To do this, start the program again, but this time, choose *Start With Full Compile* from VB's *Run* menu. VB finds the problem straight away and alerts you to its presence. The moral of this story is, be careful when changing control names. If you need to rename any controls, it would be a good idea to use VB's search-and-replace utility in the *Edit* menu to help you look for the old control names. Be careful with the “Replace All” option though, it might have unforeseen consequences. I recommend that you check the “Find Whole Word Only” option and click the Find button, making your own choices as to what should be replaced or not.

## Squashing Bugs

Let's fix the problem with the old control names – replace the existing code in `cmdConvert`'s click event handler with the following line:

```
txtToUnits = txtFromUnits * msgChosenFactor
```

That tells VB to multiply the “from” units by the multiplication factor, which the user determined by selecting a conversion type from the combo box. The result is then placed into the “to” units textbox. Notice that I've left the *Text* properties off once again, since they are not required.

**1+1=3?**

That's it – our little application should work fine now. Start the program, enter a value into the “from” units textbox and click the Convert button. A result now appears. Try choosing other conversion types from the combo box and watch the labels for the textboxes change depending on which conversion type you choose.

However, notice that in some cases, the result is very slightly out (in numerical terms). For example, if you choose the “Inches to Centimetres” conversion and then convert 10 inches into centimetres, you get 25.1999998092651 instead of 25.2! And no, it's not because you're using a Pentium. The problem is, we've mixed variable types when performing the multiplication.

### **It's just not my type**

At the moment, we're multiplying a string (the implicit *Text* property of *txtFromUnits*) by a single (*msgChosenFactor*). In reality, this isn't allowed so VB has been automatically converting the string into numerical form so that it could perform the multiplication. Likewise, VB automatically converted the result back into a string so that the result could be placed into the *txtToUnits* textbox. This is one of the cases where VB's default conversion between different variable types has gone wrong and an incorrect type has been assumed. To fix this, we need to explicitly tell VB what variable types we want to use. Replace the line of code in the click event of *cmdConvert* with the following code:

```
txtToUnits = CStr(CSng(txtFromUnits) *  
msgChosenFactor)
```

That tells VB to convert the string value in *txtFromUnits* into a *single* via *CSng*. Then, VB multiplies this by the value in *msgChosenFactor*. This works as expected since both values are *singles*. Finally, the result of that calculation is converted back into a string suitable for the *txtToUnits* textbox via *CStr*. *CStr* and *CSng* are roughly analogous to the *VAL* and *STR\$* keywords that users of other BASICs might be familiar with, except that *CStr* explicitly converts to a single whereas *VAL* does not. The code would work fine without the *CStr*, but I recommend that you always explicitly convert between types of variable to prevent problems such as the one we've just seen from occurring.

### **In Closing**

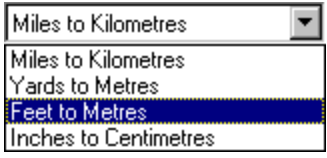
Try running the program again – you should find that everything works as planned. As an exercise, you might want to try adding your own conversions as well. If you do this, you'll need to change the dimensions of the arrays and alter the code in *Form\_Load* to accommodate the new conversion types.

Well, that's all until next time, when I'll be delving deeper into VB's innards,  
Happy tinkering,

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](mailto:nicks@miscs.com).

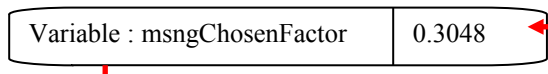
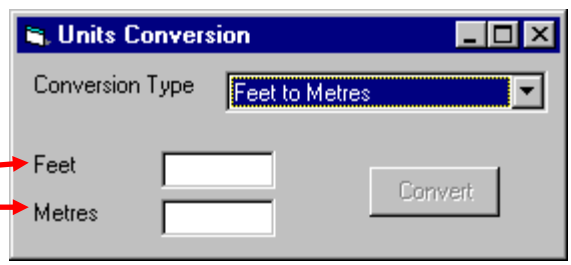
(ED: This is the large picture – it would be *really* nice if you could include it, although given its size, I find that unlikely. The filenames for the three bitmaps going from top to bottom are Image1\_ComboChoice.bmp, Image2\_FormWithChoice.bmp and Image3\_CalculationComplete.



**Step 1:** User chooses the third list item. This sets the combo box's *ListIndex* property to 2 (don't forget that list items are number from 0, not 1) and causes a *click* event to be raised.

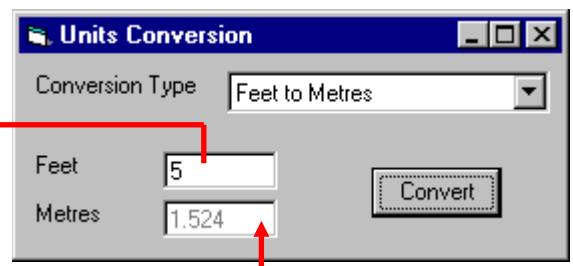
Element Number	Array Name		
	mastrToUnitNames	mastrFromUnitNames	masngMultiplicationFactors
0	Kilometres	Miles	1.6093
1	Metres	Yards	0.9144
2	Metres	Feet	0.3048
3	Centimetres	Inches	2.52

**Step 2:** The click event of the combo box looks up the elements corresponding to the *ListIndex* property (2) in the arrays, and updates the form with the appropriate values. The corresponding element (2) in array *masngMultiplicationFactors* is stored in the module-level variable *msngChosenFactor* for use later.



**Step 3:** The user enters a number (5) and clicks the *Convert* button. The value in the “from” units textbox (5) is multiplied by the value in the form-level variable *msngChosenFactor* (0.3048) to give the result (1.524) which is placed into the “to” units textbox.

$$5 \times 0.3048 = 1.524$$



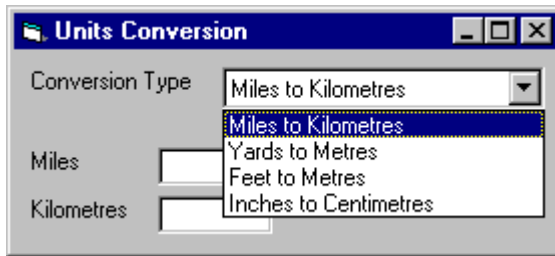
**How our application works**

(ED: Here are some alternate pictures you can use if the above won't fit)

Element Number	Array Name		
	mastrFromUnitNames	mastrToUnitNames	masngMultiplicationFactors
0	Miles	Kilometres	1.6093
1	Yards	Metres	0.9144
2	Feet	Metres	0.3048
3	Inches	Centimetres	2.52

Here are the various “from” and “to” unit names along with their associated multiplication factors. First of all, the user chooses an item from the combo box. VB sets the combo box's *ListIndex* property to reflect the number of the item that was chosen. Then, the program chooses values from the arrays whose element numbers match the *ListIndex* property of the combo box. These values are then used to change the *Caption* properties of the two label controls. The multiplication factor is stored for use later, when the actual conversion will take place.

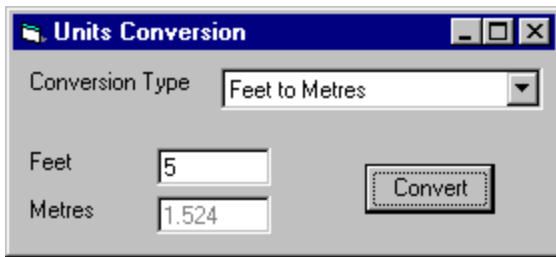
(ED: The filename for the following bitmap is Image4\_ConversionTypes.bmp)



Here is our finished application displaying the types of conversion it can handle.



(ED: The filename for this bitmap is Image2\_FormWithChoice.bmp)



Here is our finished application in action, having performed something different than the usual Inches-to-Centimetres conversion.