Visual Basic 5 Tutorial – Part 2

Last month, we created our first VB application – a simple units conversion program that converted inches into centimetres. However, it had some shortcomings, namely, that it wasn't very robust. You could easily get the application to crash (fail) by entering non-numeric characters into the txtInches textbox.

We need a mechanism to prevent our program from accepting non-numeric characters when performing the conversion. There are two approaches we can take:

1. Check for non-numeric characters in the textbox immediately before the conversion takes place. If non-numeric characters are found, display a message telling the user that only numbers should be used, and skip the conversion process

2. Prevent the user from entering non-numeric characters into the textbox in the first place

Of the two options, the second is the best choice since it is annoying to enter something, only to be told that what you've entered is invalid. It is better to prevent the invalid entry from occurring in the first place, giving a more immediate response. Identifying and rejecting bad entries made by the user of your application is something you'll frequently have to do. After all, you wouldn't be impressed if an application crashed just because you tried to enter something that it wasn't expecting. The process of checking the user's entries against what is permitted and what is not is called *validation*.

Now we know what needs to be done, but how can we intercept keystrokes and prevent them from making it to the textbox? As you might have guessed, we can use an event; the *KeyPress* event. This event is raised every time the user presses a key. Event handlers for this event get to look at what key was pressed before a control even sees it. We can use this to our advantage - if a key is pressed that we want to reject, we can just tell the control that nothing was pressed, and the invalid keystroke never makes it into the control. Since we want the txtInches textbox to reject non-numeric keypresses, we'll place some code in its KeyPress event. This code will examine each key that is pressed. If the keypress is a number key, we'll accept it, otherwise, we'll want to discard it instead. Double-click on the txtInches textbox to open the code window. VB has created a code outline for the *Change* event, since it has assumed that this is the event we're interested in. However, this isn't what we wanted – we are interested in the KeyPress event. To fix this, click on the combo box in the upper right-hand corner of the Code Window and choose the KeyPress event. VB responds, creating an outline of the code for the KeyPress event. What you *can't* see, however, is that the code that VB provided for the Change event is still there. That's because you're currently looking at the code using the *Procedure View*, which only shows you the code attached to the control/event combination shown by the combo boxes at the top of the Code Window. Look at the bottom of the Code Window, to the left of the horizontal scroll bar – there are two buttons, one resembles half a paragraph and the other looks like a full paragraph. The half-paragraph button representing the Procedure View appears to be "pushed-down",

indicating that it is the currently-selected view out of the two views available. Click the full-paragraph button to its right, which represents the *Full Module View*. VB now displays *all* the code behind any events, regardless of which controls the code belongs to. Delete the unwanted code for the Change event and then place the insertion point back between the two lines for the KeyPress event. Notice that VB changes what appears in the combo boxes depending on which event/control combination the insertion point is currently within. Now for the code itself – type in the following:

```
If Not IsNumeric(Chr(KeyAscii)) Then
   KeyAscii = 0
   Beep
End If
```

You'll notice that unlike the Click event that we've encountered previously, the KeyPress event passes us a parameter – *KeyAscii*. This represents the ASCII code of the key that has been pressed. After all, the KeyPress event wouldn't be much use if we didn't know *which* key had been pressed. For those of us that didn't know this already, ASCII is a standard code used by almost all computers, which can represent every character available by using a number between 0 and 255 (advanced users should conveniently forget Unicode). VB provides a function, *IsNumeric*, which returns true if the *string* you pass it would make a valid number, otherwise false is returned. However, KeyAscii is an *integer*, so we convert it to its character equivalent using the *Chr* function, the result of which *IsNumeric* can then act on. Experienced users might recall *CHR$* which provides the same facility in other versions of BASIC. VB doesn't require the $ on the end, so I've left it off (it looks neater). We reject the keypress by setting KeyAscii to 0, which is the ASCII equivalent of no keypress. We are effectively saying "if the key pressed wasn't a number, then reject it". The *Beep* statement makes the default Windows beep sound, to give the user a cue that something was wrong with what they did (we don't want the poor user thinking their keyboard is faulty). In the case that we want to accept the key that was pressed, we just leave KeyAscii unchanged and the textbox will see what key was pressed.

Try running the project either by clicking the "Play" button on VB's toolbar or by pressing F5. Enter some text into the txtInches textbox – the application beeps and rejects your invalid keystrokes. However, there is another twist – try deleting what you typed using the *backspace* key (that's the one above *enter*, not to be confused with the *delete* key which is below the *insert* key). Doesn't work, does it? For some arcane reason, the backspace key has an ASCII code of 8, which our code currently rejects. We need to accept this special keystroke because otherwise, the user won't be able to correct any mistakes using the backspace key. To fix this, close your application – you'll be returned to the Code Window. Change the KeyPress event handler so that it looks like this:

```
If Not IsNumeric(Chr(KeyAscii)) Then
   If KeyAscii <> 8 Then
      KeyAscii = 0
```

```
        Beep
    End If
End If
```

I've placed another *if* inside the first one, so that the keypress will only be discarded if it wasn't numeric and it wasn't the backspace key, i.e. numbers and the backspace key are allowed. More experienced users will be aware that I could have combined the two *if* tests into a single statement. However, I'm keeping things simple for the time being so please bear with me. Funnily enough, the delete key never reaches the *KeyPress* event. Nor do the cursor keys, or any of the other editing keys such as Home or End for that matter. That's why I haven't bothered testing for them. If you want to detect these keys, another event needs to be used; the *KeyDown* event. I'll leave that one for a later tutorial.

Okay, try running the application now. You should find that it works as expected. There is another problem though – try running the program, leaving the txtInches textbox blank and then clicking the cmdConvert button. The program crashes because we are multiplying by nothing, which obviously, isn't allowed. Click the End button when VB displays the error dialog, or if you clicked Debug instead, click the "Stop" button on VB's toolbar. We need something to stop the user clicking the cmdConvert button if nothing has been entered. The cmdConvert button should be initially unavailable, and then only become available when there is something to convert. We can't use the KeyPress event because not all keypresses will lead to something being entered into the txtInches textbox, for example, the delete key. What we want, is to know when the contents of the textbox change. Then, we can see if anything is in the textbox and decide whether to enable or disable the cmdConvert button as appropriate. The Change event we encountered earlier is perfect for this task. Close the Code Window and then double-click on the txtInches textbox. You'll notice that VB hasn't assumed that we wanted the Change event this time. That's because we have already entered some code for another event, so VB assumed we wanted to view that instead. Choose the Change event from the right-hand combo box and VB will create the appropriate outline code for you. The code for the KeyPress event is still in place however, since a control can have multiple event handlers, one for each type of event. Enter the following code between the lines VB has provided:

```
If txtInches.Text = "" Then
    cmdConvert.Enabled = False
Else
    cmdConvert.Enabled = True
End If
```

That should seem straightforward enough – the code now compares the contents of the txtInches textbox to nothing (the two double-quotes). If they match, i.e. nothing has been entered, the cmdConvert button is disabled, otherwise, it is enabled. The textbox is enabled and disabled using its *Enabled* property. Most VB controls have an Enabled property so that you can prevent the user from interacting with them if you wish. We want the cmdConvert button to be disabled initially, since the textbox is empty when our program starts and we can't let the user click the Convert button before something has

been entered. Change the Enabled property of the cmdConvert button to False, to ensure that it will be initially disabled.

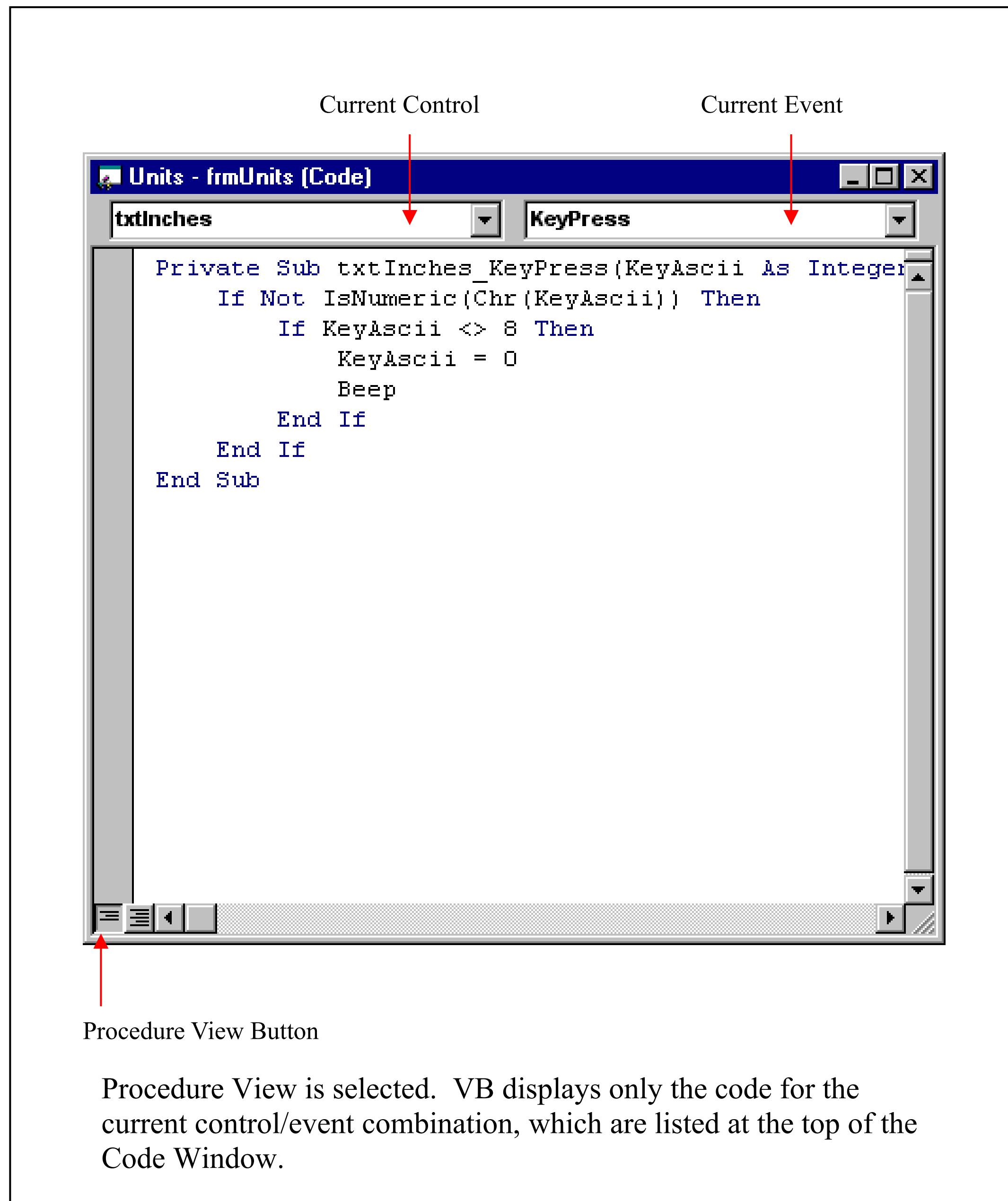
Finally, the user can overtype the result of the conversion, which isn't very professional looking. To prevent the user from changing the result, set the Enabled property of the txtCM textbox to *False*. If you try running the application now, you'll find that you cannot change the result of the conversion.

Well, that's all for this month. Sadly, there wasn't enough room to introduce any new controls. Next month, I'll introduce you to some new controls as promised. Until then, happy computing!

Nick.

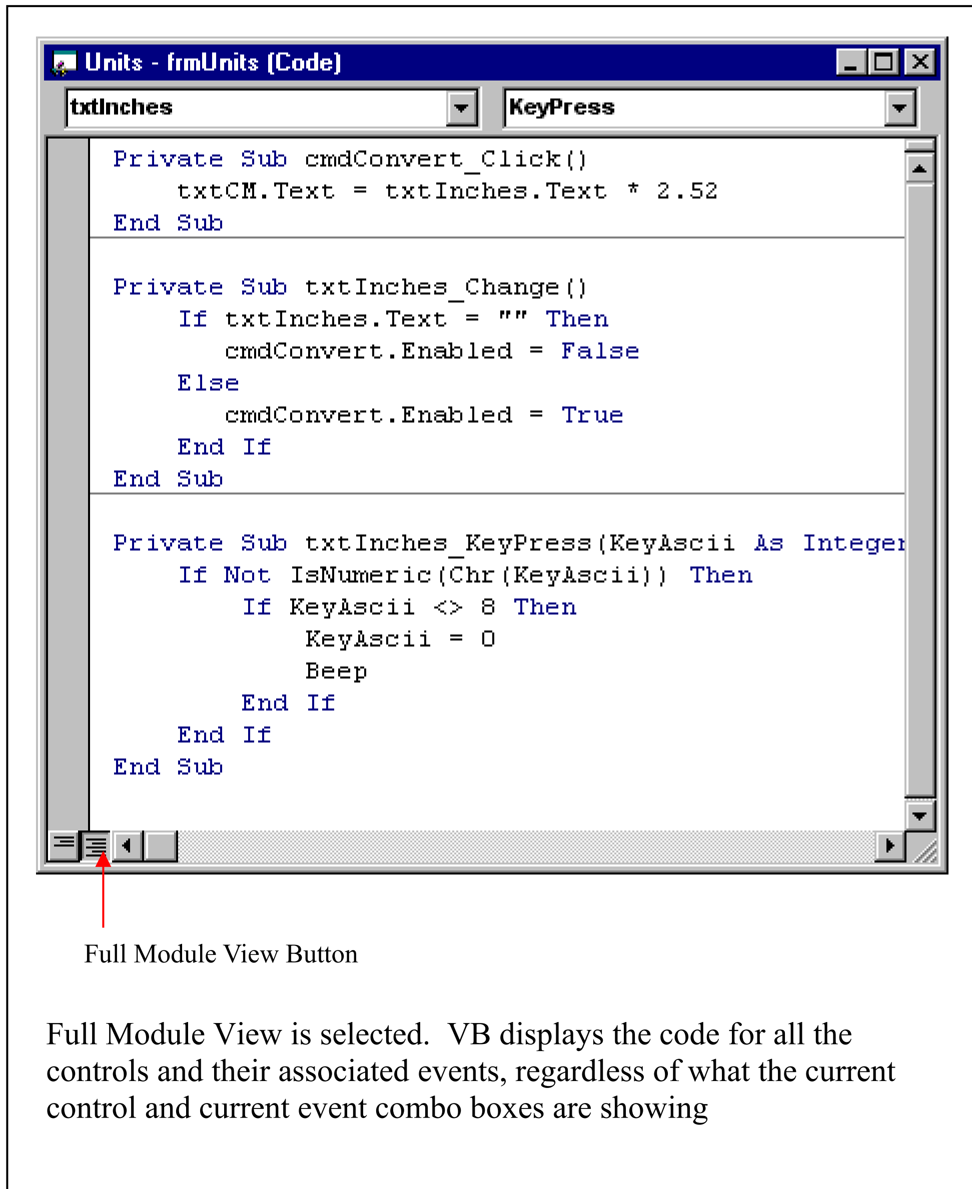
Nicholas Scott is a freelance columnist who currently works for MIS Computer Services in Northwich. Nick can be contacted via email at nicks@miscs.com.

(ED:The filename for this image is “Procedure View.bmp”)



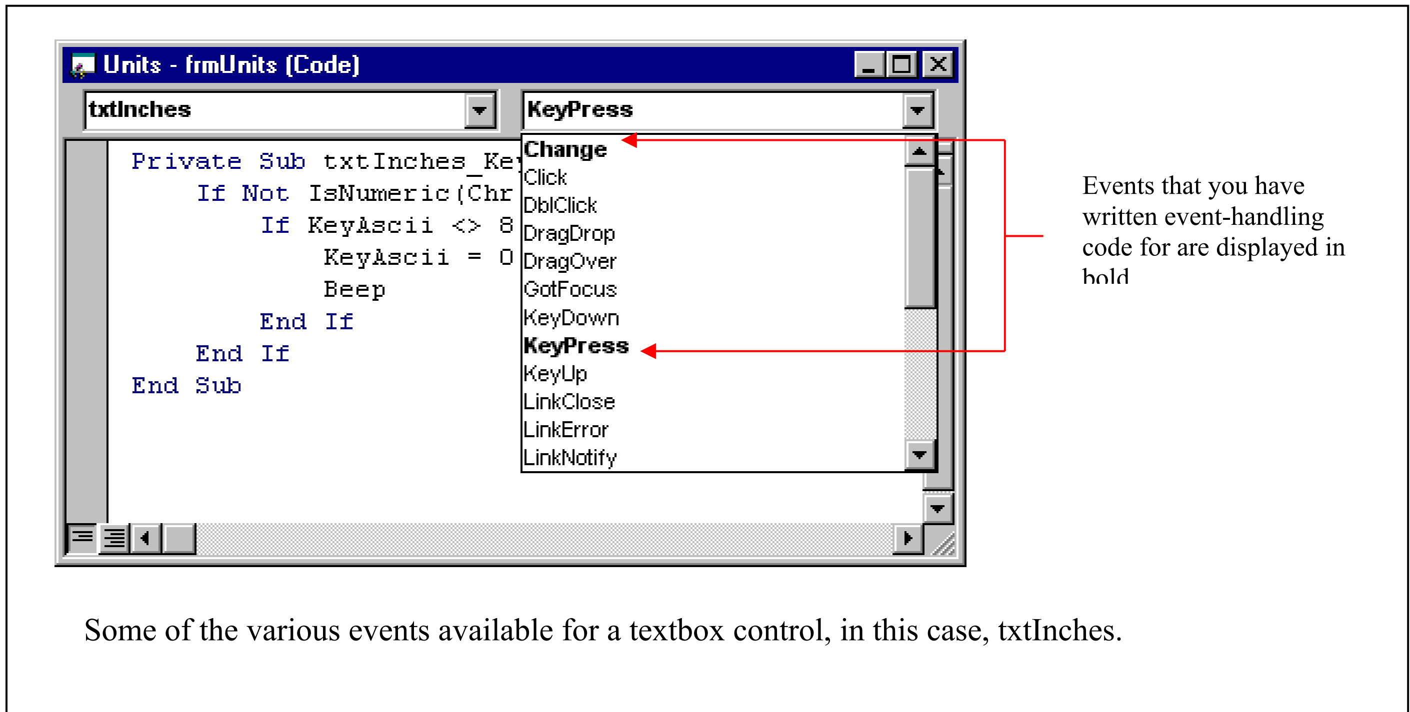
Procedure View is selected. VB displays only the code for the current control/event combination, which are listed at the top of the Code Window.

(ED: The filename for this image is "Full Module View.bmp"



Full Module View is selected. VB displays the code for all the controls and their associated events, regardless of what the current control and current event combo boxes are showing

(ED: The filename for this image is "Events Combo Box.bmp")



Some of the various events available for a textbox control, in this case, txtInches.