C# How-to 9: Get the position of the caret in a text box

Important - see the addendum to this page for the workaround to a problem with the Undo feature in Simple Editor.

One of the things it can be useful to do is to find where the caret is in a text box. The caret is the thin vertical line (usually - though it can be any shape) which indicates the position at which text will be entered into the box. Some text editors have a display of the caret position showing where the caret is at any time. How is this done?

1. Decide where the caret position will be displayed

For this demo, we will use Simple Editor, the test bed editor we are developing in C#. Traditionally the caret position is displayed in the status bar, so we need to add a new panel to the bar. I have named this 'panCaret' and given it a width of 250 pixels.

2. Get the caret position

Getting the line number is simple. The RichTextBox control has a property called SelectionStart.This is the start point of the selected text in the box in terms of a character offset from the start of the text. If there isn't any selected text, the value returned is the position of the caret, again as an offset in characters from the start of the text.

The control also has a method which returns the line the caret is in if the index position is known; this is GetLineFromCharIndex(). So getting the current line number is as simple as:

```
int line, index;

index = rtfMain.SelectionStart;
line = rtfMain.GetLineFromCharIndex(index);
```

The column is slightly trickier. Given the character index, we can get the position in the control where that character is located in the display area, in terms of X and Y pixel coordinates; this is returned in a Point structure by the method GetPositionFromCharIndex():

```
Point pt;
pt = rtfMain.GetPositionFromCharIndex(index);
```

On the face of it, this doesn't help us very much. However, there is a matching method, GetCharIndexFromPosition() which, given X and Y coordinates, will return the index of the character at that position:

```
Point pt;
int index;
index = rtfMain.GetCharIndexFromPosition(pt);
```

If the X coordinate is zero, then the character index returned is the index of the character at the start of the line the position represents. The column the caret is in is therefore the current character index minus the index of the character at the beginning of the line. Putting the whole thing together, we get:

```
private void UpdateCaretPos() {
Point pt;
int line, col, index;

// get the current line
index = rtfMain.SelectionStart;
line = rtfMain.GetLineFromCharIndex(index);

// get the caret position in pixel coordinates
pt = rtfMain.GetPositionFromCharIndex(index);
// now get the character index at the start of the line, and
// subtract from the current index to get the column
pt.X = 0;
col = index - rtfMain.GetCharIndexFromPosition(pt);
// finally, update the display in the status bar, incrementing the line and
// column values so that the first line & first character position is
// shown as "1, 1"
panCaret.Text = (++line).ToString() + ", " + (++col).ToString() ;
}
```

3. When to update the caret position

This is very simple. At first, you might think that it would be necessary to update the caret position after every event which might change it - pressing arrow keys, deleting/inserting text, clicking the mouse in the text box, etc. In fact, the RichTextBox event SelectionChanged() is fired after each of these actions, so all we have to do is call UpdateCaretPos() from this event.

There are just two refinements. The first is to decide what the caret position is when some text is selected in the box. Since text can be selected either forwards or backwards from the current position, there is no way of knowing where the caret actually would be if the selected text was then deselected - is it at the start or the end of the selected region? The easiest solution is therefore not to update the position while any text is selected, so the call in the selection changed event handler becomes:

```
private void rtfMain_SelectionChanged(object sender, System.EventArgs e) {
// update the caret position if no text is selected
if (rtfMain.SelectionLength==0) UpdateCaretPos();
}
```

Finally, the only time that the SelectionChanged event does not appear to be fired is when the text box is first displayed. At every other time, including loading a file or creating a new one, the event fires. So in the form's constructor we need to manually update the caret position display. Since the application always opens with an empty text box, the simplest way to do this is by:

```
// set the initial caret position
panCaret.Text = "1, 1";
```

which goes in the form's constructor.

4. Example code

If you try this out, you will see that the caret position is correctly updated no matter how the text changes in the box. You can download the revised Simple Editor project files from the link below. This includes the complete set of files required for this version of the program.

Download code for Simple Editor - latest version (70K)

Addendum to How-to #9 (getting the caret position in a text box)

How-to #9 showed how to use the .NET method calls in a rich text box to get the position of the caret. Unfortunately, for some reason this breaks the text box's Undo feature!

Exactly what happens is unclear, but it seems that the calls to GetPositionFromCharIndex and GetCharIndexFromPosition both cause the Undo buffer to be cleared. The result is that the CanUndo property always returns false, and the Undo function never works.

There is a workaround, but it involves using unmanaged code (the Windows API) which I didn't really want to do. However, there appears no other way round it, so here is the explanation of how it works. The technique is very well known and I wouldn't normally bother explaining it, but it does give some insight into .NET and how to call API functions from C#.

1. The API call

The API function we need is SendMessage, which sends a message to a window (and most controls are windows) requiring it to do something or return some information. There are a large number of these messages, but for the rich text box they include:

Message	Function	Maps to .NET method
EM_LINEFROMCHAR	Returns the line on which a character occurs	GetLineFromCharIndex
EM_CHARFROMPOS	Returns the character index at a specific point in the control	GetCharIndexFromPosition
EM_POSFROMCHAR	Returns the position in the control of a specific character	GetPositionFromCharIndex
EM_LINEINDEX	Returns the index of the first character on a specified line	???

The one we really need is missing! If we know the index of the first character on a given line, we can subtract this from the current caret position and get the column the caret is in on that line. Why isn't this mapped to a .NET method? Either Microsoft thought no-one was likely to need such a method, or they just plain forgot - choose your own conclusion.

So to solve the problem we need to call SendMessage with the EM_LINEINDEX parameter.

2. Declaring the function

Since SendMessage is in unmanaged code, we need to declare it accordingly. Outside of any functions in the frmMain class, we need this declaration:

```
// declare the API call
private static int EM_LINEINDEX = 0xbb;
[DllImport("user32.dll")]
extern static int SendMessage(IntPtr hwnd, int message, int wparam, int lparam);
```

This declares the constant EM_LINEINDEX and tells the compiler where to find the external function. Note the parameters to this function. The last three are all integers, but what is 'IntPtr'? Unfortunately we're back to the same old problem we faced under VB6 - how to map VB function calls to API functions. In the API, the first parameter to SendMessage is an HWND - a window handle, actually a 32-bit integer. The .NET framework returns the handles of controls which have them in the Handle property, but this is defined as an IntPtr, not an int - so we need to reflect this in the declaration.

Of the other parameters, the message parameter is set to EM_LINEINDEX, and lparam is always set to 0 with this particular message. Parameter wparam is set to the line number from which you want to get the index of the first character - but if you set it to -1, the current line (the one the caret is in) is used.

3. Calling the function

The rest is simple. The revised function in our code, UpdateCaretPos, now looks like this:

```
private void UpdateCaretPos() {
int line, col, index;
index=rtfMain.SelectionStart;
line=rtfMain.GetLineFromCharIndex(index);

col = index - SendMessage(rtfMain.Handle, EM_LINEINDEX, -1, 0);
panCaret.Text = (++line).ToString() + ", " + (++col).ToString() ;
}
```

The only difference is that we ge the caret's column using the API call rather than .NET framework calls. (It actually produces a simpler function!).

This now works as intended and does not break the Undo function. In the download, I have provided both the managed and unmanaged code, with the API call used by default. The managed code can be restored by uncommenting the appropriate lines, but then Undo will not work.

Download code for Simple Editor - latest version (72K)