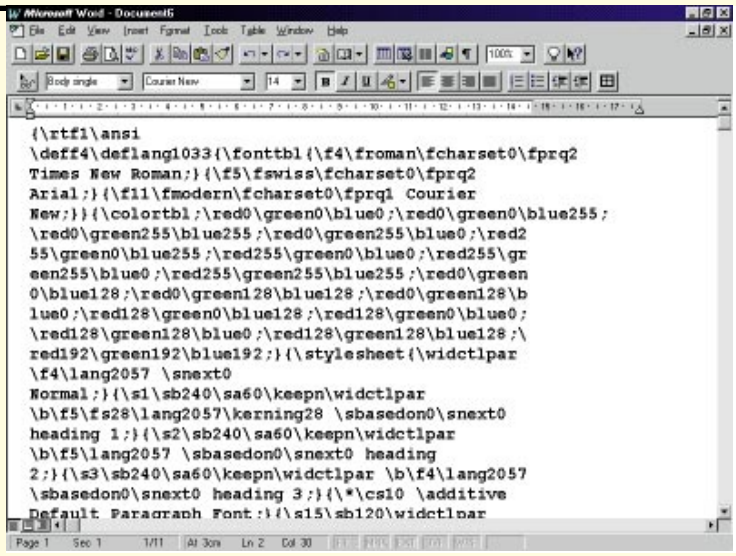
The **common** touch

Got a nice little Windows 95 or NT application under development? Including a few common controls can make all the difference, you know, and at very little cost. Plus, launching DOS programs from Delphi. With Tim Anderson.

It's always nice to get something for nothing. If you are developing for Windows 95 or Windows NT, there are a host of new features you can include in your applications for very little cost, since they are integrated into the operating system. These are the Windows 95 common controls, and you can expect to find them in most 32-bit Windows development languages including Visual Basic 4.0, Visual C++ 4.0 and the forthcoming 32-bit Delphi.

Most of the new common controls are similar to items previously available as third-party VBXs, which is bad news for VBX vendors who now have to think up new gizmos. It also presents developers with a choice: either stick with the old solutions, or adapt applications to work with the new, common equivalent.

Right *Rich Text Format in its raw form — not a pretty sight*



Below *The RichTextBox supports a reasonable range of formatting options, but cannot display justified text*

A good example is the rich text control. Once you needed HighEdit, AllText or Visual Writer to include rich text support, but now it comes as standard with VB. The add-on vendors will argue that their controls offer more features, making it possible to migrate smoothly from 16 to 32 bits by plugging in a code-compatible OCX. True; but common controls slim down your application, and as a shared resource make more efficient use of Windows. They also give your program the same look as other mainstream applications, usually considered a benefit. All this assumes that you no longer need to support 16-bit Windows.

Access to the common controls from Visual Basic 4.0 is via two OCXs. Most of the controls are in COMCTL32.OCX, while the rich text box is in RICHTX32.OCX. To show some of the possibilities, here's a look at using two of the most significant: the TreeView and RichText controls.

Windows 95 Common Controls

The following controls are supported by Windows 95 and NT 3.51 or higher, but are not available in 16-bit Windows:

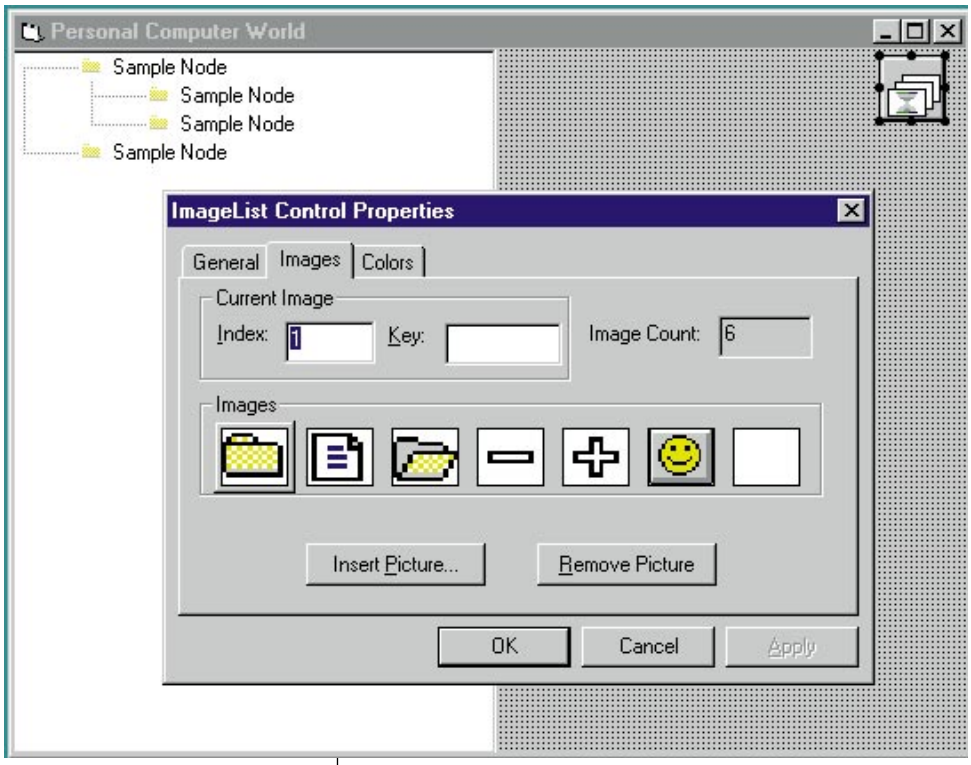
ImageList
Like the earlier PicClip control, ImageList is a way of storing images (bitmaps or icons) so that they can be used by an application without loading them individually from disk at runtime. ImageList is much easier to use than PicClip.

ListView
If you've used Explorer, you know what to expect in ListView. Like Explorer's right-hand panel, ListView enables a list of items to be displayed as large or small icons, or as a detailed list. A fourth mode, Report, allows additional text to be displayed for each item.

ProgressBar
A chunky gauge control to keep the user amused during those long operations.

Slider
As its name implies, Slider lets the user set a value with the mouse by dragging a sliding bar.

StatusBar
Your application can have a pre-built status bar divided into panels. You can write text to the panels in code, or automatically set them to display standard information such as date, time or insert/overwrite status.



The ImageList control lets you store bitmaps or icons in an application for use by other controls. In this example, the ImageList supplies bitmaps for a TreeView control

TabStrip
You can create tabbed dialogues using the TabStrip control. Because tabbed dialogues have become such an important part of many Windows interfaces, an alternative is supplied in both 16-bit and 32-bit form; the SSTab control.

BOX CONTINUES ON NEXT PAGE

Branching into TreeView
You should think of the TreeView control as a collection of nodes. Each node is a branch of the tree. To set up a TreeView, you use the Add method of the Nodes collection, the syntax for which is:

```
Nodesobject.Add(relative,
relationship, key, text, image,
selectedimage)
```

This intimidating parameter list is not really so bad: it tells the TreeView control where the new node fits in the hierarchy, provides an unique identifying key and, optionally, an image to display by the mode. For instance, you might use TreeView to construct a multimedia viewer: the book title is the root of the tree; chapters are the next branches down; sections are below each chapter.

Fig 1 shows a parameter list in which the image parameters are unused. When included, they refer to the index of an ImageList control with the selected image parameter, making it possible to change the image when the node is selected.

Once the tree is set up, the chances are you will want something to happen as the user navigates through the tree. The most useful event is NodeClick, which provides the current Node object. For example, the following will place the text of the current node into a RichTextBox:

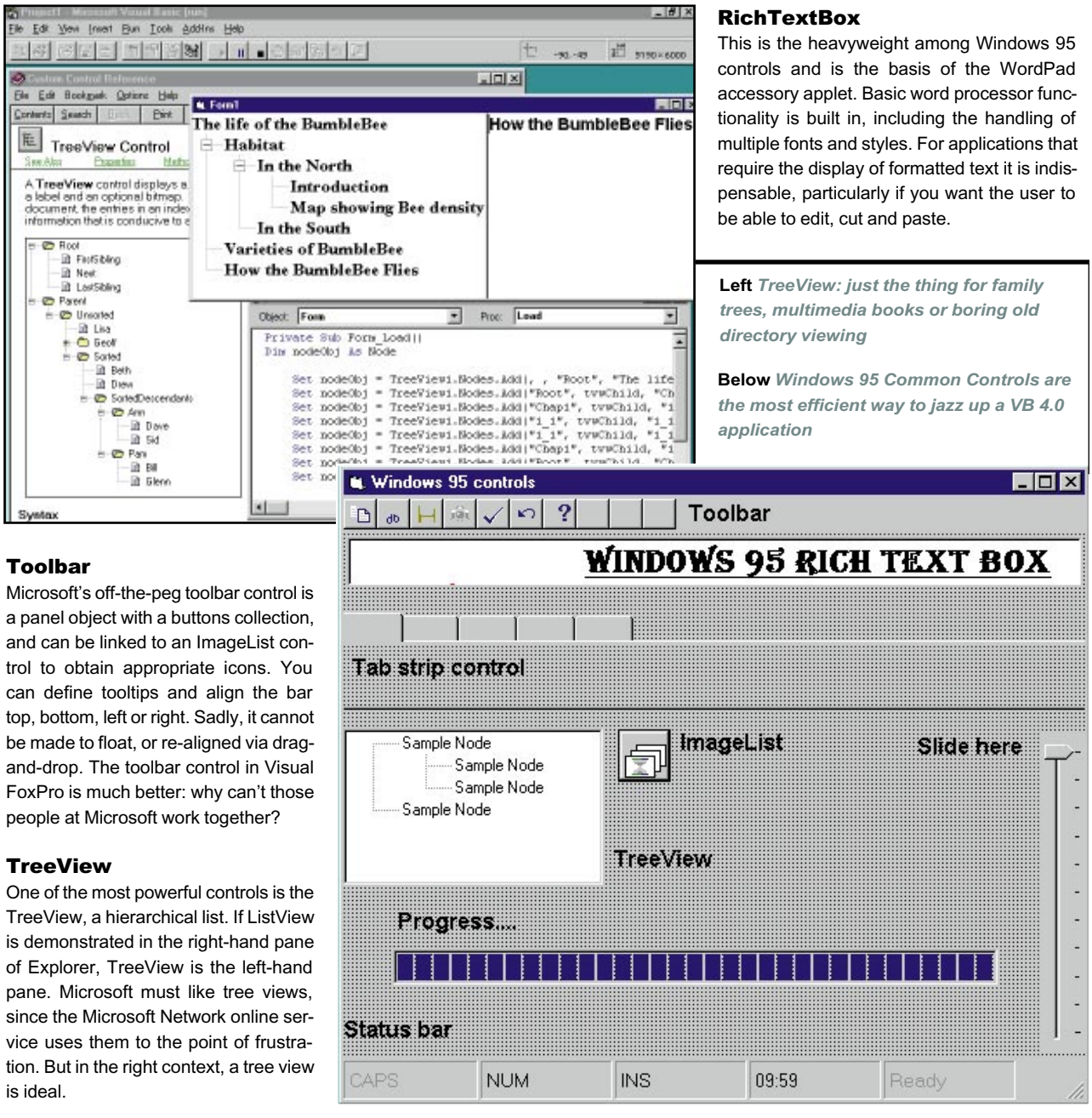
```
Private Sub TreeView1_NodeClick
(ByVal Node As Node)
RichTextBox1.Text = Node.Text
End Sub
```

To develop the idea further, one approach would be to store rich text documents in an Access database. The Key for each node could identify a record in the database, and in the NodeClick event you could write code to display the text, and hey presto! a multimedia document and viewer for the price of very little code.

Fig 1 Parameter list

```
Dim nodeObj As Node
Set nodeObj = TreeView1.Nodes.Add(, , "Root", "The life of the BumbleBee")
Set nodeObj = TreeView1.Nodes.Add("Root", tvwChild, "Chap1", "Habitat")
Set nodeObj = TreeView1.Nodes.Add("Chap1", tvwChild, "1_1", "In the North")
Set nodeObj = TreeView1.Nodes.Add("Root", tvwChild, "Chap2", "Varieties of BumbleBee")
Set nodeObj = TreeView1.Nodes.Add("Root", tvwChild, "Chap3", "How the BumbleBee Flies")
```

More Windows 95 Common Controls



Left *TreeView: just the thing for family trees, multimedia books or boring old directory viewing*

Below *Windows 95 Common Controls are the most efficient way to jazz up a VB 4.0 application*

RichTextBox

This is the heavyweight among Windows 95 controls and is the basis of the WordPad accessory applet. Basic word processor functionality is built in, including the handling of multiple fonts and styles. For applications that require the display of formatted text it is indispensable, particularly if you want the user to be able to edit, cut and paste.

Toolbar

Microsoft's off-the-peg toolbar control is a panel object with a buttons collection, and can be linked to an ImageList control to obtain appropriate icons. You can define tooltips and align the bar top, bottom, left or right. Sadly, it cannot be made to float, or re-aligned via drag-and-drop. The toolbar control in Visual FoxPro is much better: why can't those people at Microsoft work together?

TreeView

One of the most powerful controls is the TreeView, a hierarchical list. If ListView is demonstrated in the right-hand pane of Explorer, TreeView is the left-hand pane. Microsoft must like tree views, since the Microsoft Network online service uses them to the point of frustration. But in the right context, a tree view is ideal.

Get RichText quick

The RichTextBox is a superset of the standard edit control. You should never use a Rich Text control where the standard item will do, since it consumes more resources. With a standard text box, properties like FontName and FontSize affect all the text in the control. The RichTextBox has SelFontName and SelFontSize properties instead and these alter either any selected text, or the next characters typed if no text is selected. That makes it easy to format text in code, or from a toolbar or menu.

You can load and save files with the RichTextBox either in plain text or RTF format. Clipboard support is automatic. There's also a Find method which searches for a given character string. Printing is carried out using the SelPrint method. SelPrint will output selected text, or the whole contents of the control if no text is selected. It does not print directly but needs a handle to a device context. The simplest approach is to use the VB Printer object which has an hDC property. Note that this is not valid until something has been printed, so you need to print an empty string before calling SelPrint, with code like this:

```
Printer.Print ""
RichTextBox1.SelPrint (Printer.hDC)
```

The RichTextBox is data-aware. Simply binding it to a memo field gives you the basics of a book viewer or document management system as it automatically reads and writes RTF documents to the database. Overall, this is an excellent control, although as it lacks the ability to justify text, display pictures or include OLE objects, there is still space for third-party rivals.

Zipping into Delphi

One of the most enduring DOS programs must be PKZIP. Adam H writes:

"I'm brand new to Delphi but getting along (I think). I have one question that is

really bugging me. Can you launch other applications from an app created with Delphi, and if so, how?

I wish to create a simple application with two buttons on — BACKUP and RESTORE. When a button is pressed, it will run the PKZIP program and compress a directory onto a floppy, or vice versa.

Also, can you trap the messages issued by the PKZIP program and place them into the Delphi app?"

You can launch applications from Delphi using the API functions WinExec or LoadModule. For example, the following runs the Windows calculator:

```
WinExec('CALC.EXE', SW_SHOW);
```

In Visual Basic you can do the same thing with the Shell command, but this does not achieve what Alan requires. PKZIP is a DOS program, so to run it in Windows will open a DOS window, execute the program, and leave the window on the screen. You can improve on that, for example by creating a .PIF file for PKZIP, specifying that the DOS window closes on termination, and using the SW_HIDE parameter:

```
winExec('PKZIP.PIF      A:\BACKUP.ZIP
C:\DOCS\*.DOC', SW_HIDE);
```

That works well if there are no problems, but what about error handling? What if PKZIP produces one of its "y/n" prompts and waits forever in a hidden window for an answer? At this point, things get nasty. Alan wants to trap PKZIP's messages, but even if he uses DOS functions to redirect PKZIP's output to a file for parsing later, there will still be difficulties with prompts that need a user response. We are back to the untidy DOS window solution. Another problem is that WinExec is asynchronous: the Delphi or VB application will continue to execute at the same time PKZIP is running. The program would need to enter a loop, calling the API function GetModuleUsage with the instanceID returned by WinExec, until it returns zero to indicate that the program has terminated.

You can do a better job of running DOS programs from Windows by writing a Virtual Device Driver (VxD) to intercept DOS stdout and stderr — not exactly visual programming. The easier solution in this case is to use a compression library designed to be integrated into applications. There are several to choose from, including Microhelp's Compression Plus and PkWare's Data Compression Library.

What is Rich Text Format?

Rich Text Format (RTF) is Microsoft's format for transferring formatted text between programs. It includes codes to identify fonts and styles. RTF uses the backslash to begin control words, and brace characters to identify groups of text. For example, the code "\b" means "bold", and "\b0" means "turn bold off". Therefore, you can embolden a word in an RTF document like this:

```
Here is a \b bold \b0 word in RTF
```

Because an RTF file generally uses only plain text, it is easy to transfer between different applications, or across different platforms like PC to Apple Macintosh. Usually it is more reliable than using conversion routines that work on word processor formats like those used by Word, WordPerfect or Ami Pro. RTF is particularly important in Windows, since it is the standard clipboard format for transferring formatted text.

The RTF standard has caught on well and every self-respecting word processor or DTP application supports it. The snag is that Microsoft controls the format and tends to add zillions of new control words with every new version of Word. This has caused problems for the industry. An example of this is the Windows help compiler which is meant to work with RTF documents from any source, but has a mysterious preference for those produced by Word. For most other purposes, though, older subsets of the full RTF standard are quite adequate. RTF is documented by Microsoft, and its description, together with code for a simple RTF reader, can be found on the Developer Network CD.

Next month

Just arrived is Crystal Reports Professional 4.5, and new OCX versions of First Impression and Visual Speller. Look for a report in the next issue.

PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted via *PCW* at the usual address, or on **freer@cix.compulink.co.uk**