Stabilising influence

Is Visual Basic 5.0 flaky? Tim Anderson offers some tips on preventing problems. He also tries out BoundsChecker for Delphi and answers your visual programming queries.

Several readers have reported problems with Visual Basic 5.0. Tim Gathercole writes: "*I upgraded to VB5 Professional about three weeks ago and the program is proving to be highly unstable when working with larger projects. Four of my projects, which worked fine under versions 2.0, 3.0 and 4.0, now either crash version 5.0 at runtime or work in runtime but crash when compiled. I got the same result on both a Pentium and a Pentium Pro machine, each with 32Mb of RAM.*" He goes on to enquire about a VB5 to VB4 converter, in order to get his applications running again.

If anyone doubts that VB 5.0 is less stable than it should be, try this: select two or more controls on a form (anything with a font property will do). Have the properties window floating and the environment set to MDI (the default). Then try to change the font. VB responds: "This program has performed an illegal operation and will be shut down".

While this font problem is a reproducible bug, other faults seem to be annoyingly unpredictable. Some users experience constant crashes, while others find VB 5.0 stable and reliable. Native code compilation can be problematic. Microsoft claims a bug-fix release is on the way, but why all the problems? There seems to be a variety of reasons: the faster forms engine is fussier about the video card and drivers than previous versions; native compilation introduces a new layer of complexity; and VB's dependence on ActiveX makes it susceptible to any problems with OLE or the system registry. Here are some tips for those struggling to get VB 5.0 working:

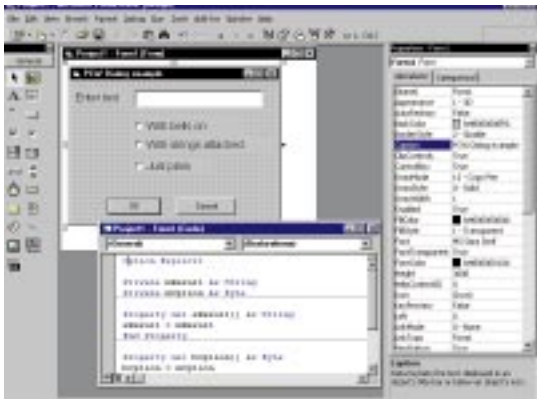
- Check "When a program starts — Save Changes" in the Options-Environment dialog.
- Do a fresh installation of Windows and VB, preferably onto a newly-formatted hard disk. This brute force approach corrects two common problems: mismatched system DLLs and/or a corrupt registry.
- Double-check API calls and any calls to external DLLs.
- Check your current memory and add more if possible. Thirty-two megabytes is good.
- VB 5.0 sometimes corrupts memory when forms with many controls are loaded. Removing a form from a project and adding it back can help.
- Deselect toolbars in the view-toolbars-customise menu.
- Remove add-ins.
- Switch to standard Windows video drivers.

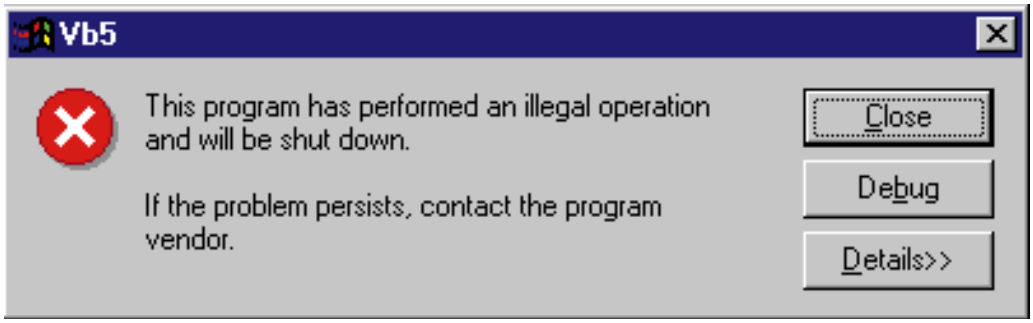
Dealing with dialogs

Tim George is using VB4 and tells me: "*I would like to be able to create a dialog box which returns a single value, much like the inputbox function but with extra functionality. My problem is that I wish to pass data to and from the dialog box without resorting to global variables, but using some kind of parameter list?*"

In the days of Visual Basic 3.0, global variables were hard to avoid. The only other option was to keep the form in memory by using the Hide method rather than unload. Then you could refer to the values of the form's controls after the user has closed the dialog. A better approach in VB 4.0 or higher is to use custom properties.

In the General section of a form module,





Above A good way to do dialogs in Visual Basic is by using property procedures. The secret is to remember that in VB 4.0 and higher, forms are like class modules Left Visual Basic 5.0 has many great features, but is spoilt by instability during development and at runtime

p280 ➢

Taking a look at books

■ **Using Delphi 3.0**
by Todd Miller and David Powell
Delphi's weakest point is its documentation. The online help is notorious for broken links and skimpy examples, while the printed manuals leave important subjects virtually untouched. That opens the way for third-party alternatives like this.

The book's scope is impressive. In just over 1,000 pages it runs from introductory basics to the mysteries of "thunking", calling 16-bit DLL functions from 32-bit code or vice-versa. The level is suitable for anyone with some programming experience. There is balanced coverage of neglected topics like threads, OLE automation, database programming and creating ISAPI DLLs for web server applications.

It is weak on techniques for object-oriented programming. The bundled CD includes example code, product demos, and four strangely unrelated Que titles in HTML format. Recommended.

■ **Presenting JavaBeans**
by Michael Morrison
Visual Basic and Delphi programmers should take a keen interest in JavaBeans. Beans are reusable visual or non-visual components that expose properties and handle events, just like ActiveX controls in VB or components from Delphi's Visual Component Library.

Once suitable visual tools arrive, JavaBeans will make rapid application development easier and more effective. This short title explains the JavaBeans API, and gives four complete examples using Sun's Bean Development Kit. With a warm, good-humoured writing style, the author explains key concepts like properties, introspection, event handling and persistence. Some knowledge of Java is assumed.

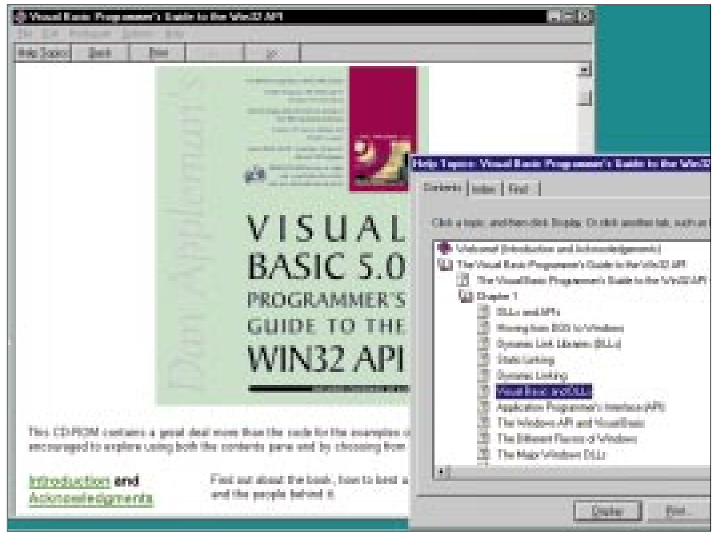
It is a basic introduction, useful for anyone wanting to get up to speed on what JavaBeans is all about.

■ **Visual Basic 5.0 Programmer's Guide to the Win32 API** by Dan Appleman
You have to admire an author who writes in his foreword: "If you already own the original, the changes do not justify the price of a new book."

The book in question is Appleman's classic guide to calling the Windows API from Visual Basic. The previous edition was extensively revised to cover the transition to 32-bit Windows, but this time around the API is essentially the same, so fewer changes are necessary.

Actually, Appleman is right. The changes are not enough to justify a new purchase unless you have to have the latest of everything. The new edition is revised for Visual Basic 5.0 although version 4.0 is still extensively covered for the sake of its 16-bit compatibility. The frequent plugs for products from Desaware, the author's company, are a little tiresome, though.

The book remains a superb reference for those who need to go beyond Visual Basic's built-in functionality. For advanced work with menus or fonts, for example, it is near-essential. The accompanying CD has a slightly expanded version of the book in Windows help format, along with Desaware's API class library.



The CD which comes with Appleman's *Guide to the Win32 API* has the whole book as a Windows Help file. This is easier to navigate and search than the more common HTML or PDF electronic formats

declare private variables for the dialog's values. Next, create property procedures to access the variables. In your dialog's OK procedure, write the values from the dialog controls back to the private variables. Finally, you can create a ShowDialog method which takes parameters to initialise the dialog values.

When you have finished with the dialog, use SET MyForm = Nothing to clear the form module from memory. Here is some example code:

```
'Code for the form module

Private mResult As String
Property Get sResult() As String
sResult = mResult
End Property
Sub ShowDialog(sParm As String)
mResult = sParm
Me.Show 1 ' show form modally
End Sub
Private Sub Form_Load()
Text1.Text = mResult
End Sub
Private Sub cbOK_Click()
mResult = Text1.Text
Unload Me
End Sub
' code to use the dialog
MyForm.ShowDialog("MyParameter")
MyResult = MyForm.sResult
Set MyForm = Nothing ' clears form
 module from memory
```

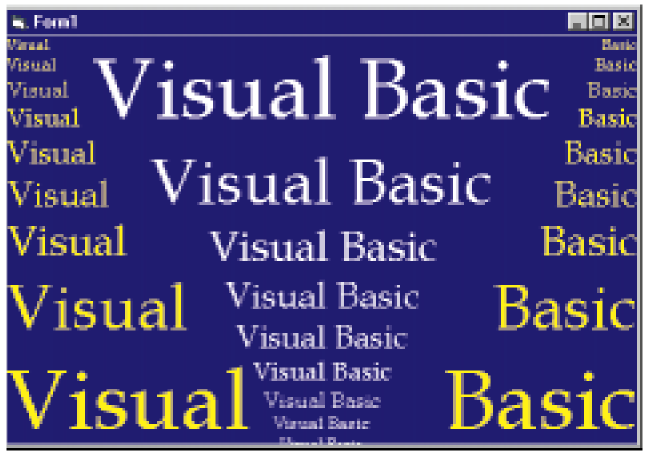
Poster poser

Liam McAllister is working with fonts in Visual Basic. *"I am trying to write a program to print posters of various sizes. When I want to print out the poster, I take a bitmap copy of a form with labels and send it to my printer, stretching/shrinking (using API functions) as necessary. This works fine up to about A4 size but, as you can imagine, when stretched to this size the text looks very jaggy and rough. Also, the text I use on the labels is varying in the font style and the text entered: I would like the text entered to fit into boxes of a predetermined size."*

Sending text to the printer as a bitmap is not ideal. It is better to use the Print method of the Printer object, as this will allow Windows to scale the font correctly to the resolution of the output device. The disadvantage, though, is that you need to work harder to position text and graphics elements correctly, using the printer object's CurrentX and CurrentY properties.

To determine how much space text will take up when displayed in a label, use the TextWidth method. TextWidth takes a string parameter and returns the width of the text when printed in the current font and size. Unfortunately, labels do not have a TextWidth method, so one possibility would be to use a hidden picture box, adjust its font as needed, and measure text with TextWidth to discover how much will fit into a label which has the same font.

Whatever you do, do not place numerous picture boxes on a form to use as fancy label controls, as this is wasteful of Windows resources. Labels are better or, better still, print directly to the form using its print method. This approach has advantages when you need paper output as well, as you can easily adapt the code to work on the printer object instead of the form.



Left **By writing code to print directly to a form, and using TextWidth to measure strings, you can obtain good-looking font effects**

Book search

People often ask me for book recommendations. Geraint Preston asks: *"Could you recommend the best book from which to learn C++? I've worked my way through* A Book on C *and I'm keen to go on to the next stage. Could you also recommend (though my need is less pressing) the best book from which to learn Visual Basic? I bought Microsoft VB and C++ together at a special price."*

The problem with the C++ question is that there are at least four separate skills to learn. First, there are the bare bones of the language, and there are plenty of tutorials available. Next there is the question of object-orientated programming. Third, a tool like Visual C++ has a complex interface and a product-specific book is helpful. Fourth, most C++ programmers work with a class library which needs learning in its own right. An additional factor for Windows programmers is that you need to know how Windows itself fits together, which means familiarity with the Windows API. Since Geraint has Visual C++, a good choice would be *Inside Visual C++* by David Kruglinski (Microsoft Press).

The tutorials included with Visual C++ are also good. Be warned that these resources are specific to Windows, the Microsoft Foundation Classes and Visual C++. If you want general C++ skills, they may be positively unhelpful.

Visual Basic is easier to learn. Another advantage is that the supplied manuals are actually very good. If your version came without printed manuals, you can buy them separately as Microsoft Press titles. Once you have digested the official *Programmer's Guide*, you will be ready to tackle a more specialist title depending on which aspect you want to focus on. Books are regularly reviewed in this Hands On section and past columns in this section are included on our *PCW* cover-disc.

Name that user

P Blomfield is using VB 4.0. He asks: *"On a network, how do you find the machine name? How do you find the username of the current user logged in?"* This information is provided by the Windows API. The relevant functions are GetUserName and GetComputerName. If you copy the declaration from VB's API viewer, you can then call the functions (Fig 1).

Fig 1 Finding the user

```
Dim sBuffer As String
Dim lRetVal As Long
Dim lLength As Long

sBuffer = String(32, " ")
lLength = 31
lRetVal = GetComputerName(sBuffer, lLength)
Label1.Caption = "Computer name: " & Left(sBuffer, lLength)
sBuffer = String(32, " ")
lLength = 31
lRetVal = GetUserName(sBuffer, lLength)
Label2.Caption = "User name: " & Left(sBuffer,lLength)
```

BoundsChecker comes to Delphi

BoundsChecker is the flagship product of NuMega Technologies. Its purpose is to track down bugs which are otherwise hard to catch. Until now it has been for C or C++ developers only, but version 5.0 gives Delphi applications the same treatment.

Boundschecker installs itself into the Tools menu of the Delphi 2.0 IDE. A patch to work with Delphi 3.0 should be available from NuMega's website by the time you read this. Delphi 1.0 is not supported.

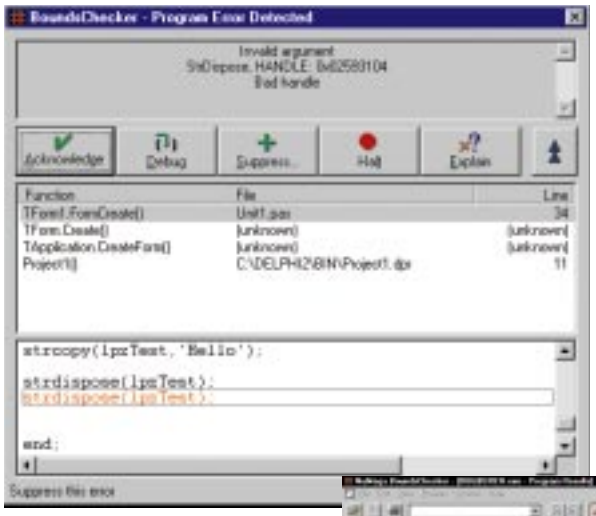
The natural question to ask is: what can BoundsChecker do that Delphi's own debugger cannot? One of the issues is detecting memory leaks. For example, your program might include the following code:

```
var
nullstring: pchar;
begin
nullstring := strAlloc(50);
strcopy(nullstring,'Never freed');
end;
```

Delphi's debugger will not show any fault and the program will run fine. The problem is that memory is allocated but never freed. When BoundsChecker is active, the leak is reported in an error window after the program runs. Double-clicking the error takes you to the point in the source where the memory was allocated.

Another type of leak occurs when objects are created in code but not freed. For example, you might have a listbox which contains a list of music CDs. Using the AddObject method, you could associate a TStringList object with each item in the list box, perhaps to store a track listing. When the listbox is destroyed, Delphi will automatically free the items in the list but

p283 ➢



Left This pop-up dialog intercepts an error and offers several options for dealing with it

Below BoundsChecker shows the type of error, source code when available, call stack, and online help explaining how to fix it

not the associated StringList objects. Running this through BoundsChecker reveals the leak, for example, "32 bytes allocated by ReallocMem in TStringList.SetCapacity". The solution is to write code that frees the StringList objects before the listbox is destroyed.

BoundsChecker illustrates the point that Delphi is not as safe an environment as Visual Basic or even Java. These languages have a feature called garbage collection which frees objects once no valid reference to them exists. Advanced Delphi programmers need to do their own memory management to some extent. In particular, working with the Windows API inevitably means using pointers, the most common source of memory errors.

NuMega's BoundsChecker reports API and OLE errors as well as memory leaks. It also has an event reporting feature. When enabled, this collects all the Windows API calls, parameters and messages your program sends and receives. The mass of resulting information does make it hard to track down problems, but with patience this can reveal places where code is not working or efficiency can be improved.

This is a specialised tool. While it is great for finding memory problems, it will not help you find logic errors in your code. It is most useful for more advanced programmers with large applications to debug, and in these situations should soon repay its purchase price. Its integration with Delphi is impressive, or will be when the Delphi 3.0 patch is available.

Note that BoundsChecker for Delphi is not quite as capable as the C++ version. The latest Visual C++ edition uses a newer technique called FinalCheck which can find an additional set of memory and pointer errors. Even so, BoundsChecker is an excellent resource for Delphi developers.

And finally…

An unfortunate mix-up resulted in the June 1997 issue Hands On Visual Programming feature not being printed last month — in its place, an older version appeared. The actual June issue column can be found at www.cix.co.uk/~tim-anderson. Topics include Visual Basic in Visio, VB 5 subclassing, and exploiting units in Delphi.

PCW Contacts

Tim Anderson welcomes your Visual Programming tips and queries. He can be contacted at the usual *PCW* address or at **visual@pcw.co.uk**.

BoundsChecker 5.0, Delphi Edition, costs £345 + VAT from Grey Matter 01364 654100. Further information at www.numega.com.

The following books are available from Computer Manuals 0121 706 6000:

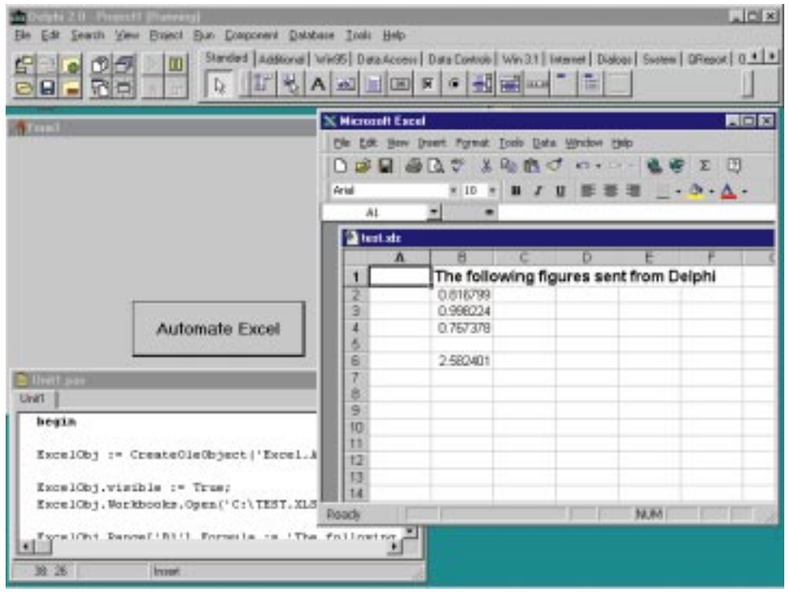
- *Visual Basic 5.0 Programmer's Guide to the Win32 API* by Dan Appleman, Ziff-Davis Press. Book and CD £54.95 (inc VAT). Further information at www.desaware.com.
- *Using Delphi 3.0* by Todd Miller, David Powell and others, Que. Book and CD £46.99 (inc VAT).
- *Presenting JavaBeans* by Michael Morrison, Que. Book and CD £32.95 (inc VAT).

The Outlook is variable

Tim Anderson finds Outlook, the latest Visual Basic-enabled Office 97 application, to be a powerful but frustrating solution. He gives his views about programming and creating forms.

Cedric Maddox writes: "*I'm trying to get to grips with Delphi. Your review of the Appleman Guide was very interesting and I would like a similar guide for Delphi: can you recommend one? If there isn't a Delphi guide available, would the Appleman Guide for VB help with Delphi API calls?*"

Much of Daniel Appleman's guide to the Win32 API (Ziff-Davis) would be useful to Delphi developers since it is, after all, the same API. I hesitate to recommend it though, because large parts are specific to Visual Basic. Calling the Windows API is actually easier in Delphi than in Visual Basic: partly because the language is a better fit, with direct support for pointers and easy handling of Windows messages; and partly because Delphi's developers have helpfully defined the types, function and procedure headers for you.



Get the syntax right and automating Excel is a powerful way of extending a Delphi application (*see "Delphi and Excel 97", below*)

Often, you can use an API function as if it were native Pascal. But although calling the API is easy, understanding how it works is another matter. More advanced Delphi titles like *Delphi 2.0 Unleashed* or *Delphi 2 Developer's Guide*, both from Sams, contain helpful API tips. The official Microsoft reference is essential, and an online version comes with Delphi. Finally, Charles Petzold's *Programming Windows 95* (Microsoft Press) is useful not as a reference title, but as an explanation of how Windows works behind the scenes.

Unfortunately, if you want to get deeply into the Windows API, you have to put up with reference material aimed at C and C++ programmers, since Windows itself is mainly written in these languages.

Delphi and Excel 97

Another reader's problem is: "*According to my testing, there is a problem with Delphi 2 and OLE-automation using the version of Excel delivered with Office 97. Specifically, I can start Excel via OLE-automation and do some things (like ExcelApp.Visible := True), but if I try to access a range, for instance, it just crashes with an EOLESysError. Exactly the same test program works fine if I uninstall Office 97 and use Office 95 instead.*"

I suspect this is a bracket problem. The code in Fig 1, when used with Excel 97, fails with a "Member not found" message. The solution is to replace the last line with:

```
RangeObj := ExcelObj.
Range['B2:B4'];
```

Rich text problems

Gavin Docherty writes: "*For about three months I have been trying to paste bitmaps, metafiles and OLE objects into the standard richtextbox control found in the 32-bit common control DLL but without success. Then, to my amazement, you covered the subject in your May column and gave code examples for VB, but I couldn't get it to work. What have I done wrong?*"

Unfortunately, the news is not as good as I thought. The code printed in May's issue was tried and tested with Visual Basic 4.0. But at some point, some application or other had installed a later version of the RICHTX32.OCX component which makes it work, complete with an updated help file. This later version is required in order to work with pictures.

Programming Outlook 97

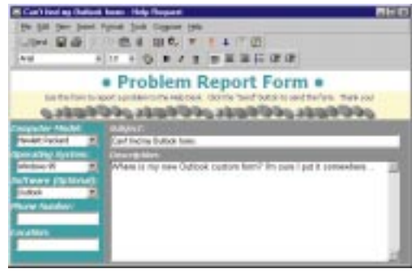
Outlook is an excellent starting point for an Office 97 application. Most people need an email reader and an address book or contact manager, and Outlook does both. The natural next step is to add functionality. For example, Outlook's contact menu already has an option to start a new letter to the current contact.

You might want to add new options, perhaps a choice of several standard letters. Getting more ambitious, you could pull in other information such as account information or product preferences. Another scenario could find you placing an order for a contact you have just called. Many things are possible since Outlook has a forms designer and a scripting language, but this is VB Script and not the powerful Visual Basic for Applications. VB Script is the cut-down version of Visual

p299 ➢

Book Reviews

■ **Building Applications with Outlook 97**
£37.49 (book and CD)
Microsoft Press



The Help Desk application is explained in *Building Outlook Applications*

Outlook is a strange combination of elegant simplicity and arcane complexity. The Outlook bar is delightful, with easy access to network and internet mail, appointments, contacts and to-do list. With its own form designer and a version of Visual Basic, it seems the ideal platform for building groupware applications. Start developing, though, and Outlook shows its other face. Want to save a form? There are three ways to do it, claims the online help. Did you want the open folder, the file or the forms library? And if the latter, did you want the personal forms library, the folder forms library, or the organisation forms library?

The other problem is that Microsoft seems determined to disguise the close relationship between Outlook and Exchange. This book is a case in point. The cover wording refers several times to groupware but never to Exchange. Without Exchange, though, Outlook is only suitable for groups of one.

If you do have Exchange Server and want to develop with Outlook, this title is all-but essential. It describes Outlook's main elements and explains how to use the form designer and Visual Basic Script. It concludes with a step-by-step guide to creating three sample applications: the first is a form for requesting business cards, the second a help desk application, and the third tracks document production. This last is the most interesting since it links to an Access database using Data Access Objects. It's a useful guide, and really should have been part of the Office 97 Developer Edition. The only other caveat is that you will need a lot more help than this when it comes to managing the back-end of an Outlook application, Exchange Server itself.

■ **Using Visual Basic 5.0**
by Mike McKelvy, Ronald Martinsen and Jeff Webb — £36.99, Que

Smartly published to coincide with the release of Visual Basic 5.0, this title in Que's classic "*Using*" series begins right at the beginning, with topics like what is a program? And what is a variable? By the end of its 950 pages it is tackling API programming, callback functions and remote automation servers. The result is a comprehensive book, but lacking in sparkle.

There is too much here for real beginners, while experienced VB developers migrating to version 5.0 will find themselves skipping large chunks of the material. It has more the style of a manual than a real-world developer's book. Because Visual Basic is now such a large product, this comprehensive approach means little depth on any individual subject.

This is a good buy if you do not have printed documentation, with clear, thorough explanations covering every aspect of Visual Basic. Others will be better served by one of the many more specialist Visual Basic titles now available.

■ **Instant Visual Basic 5.0 ActiveX Control Creation** — £27.49, Wrox Press

Sporting no less than seven authors, this title covers the hottest new feature of Visual Basic 5.0: the ability to create ActiveX controls. It mainly covers the Control Creation Edition, although it will be equally useful to owners of the full version of Visual Basic. It is aimed at experienced VB developers.

After an introductory section, the main part of the book takes you, blow-by-blow, through developing several example controls, including an aggregate control, a data-bound control, and a user-drawn control. An aggregate control is one that contains several other controls, while the term "user-drawn" describes a control whose visual display is handled entirely by the program.

Despite the book's multiple authorship, the style is clear and consistent. Creating ActiveX controls presents many new issues for VB developers and this is a helpful and detailed guide. It would be better still if more space were given to the design issues behind component programming as opposed to just the mechanics of how to do it. Other vital topics are version control and web security, neither of which are given sufficient coverage. This does not detract from the high quality of the subject matter included: recommended.

● *The above books are available from Computer Manuals on 0121 706 6000.*

Fig 1: A problem with brackets

```
var
ExcelObj: variant;
RangeObj: variant;

begin

ExcelObj := CreateOleObject('Excel.Application');
ExcelObj.visible := True;
ExcelObj.Workbooks.Open('C:\TEST.XLS');
RangeObj := ExcelObj.Range('B2:B4');
...
```

Basic first used in Internet Explorer.

Outlook is a handy contact manager but its database is a simple flat-file affair which is not suitable as the main data store for a business. The key then is to integrate Outlook with other data sources. Since VB Script has neither built-in database features nor the GetObject or CreateObject functions needed for programming COM objects, it does not, at first, seem promising.

By a roundabout route, though, it does have those functions. Outlook's Application object has a CreateObject method that opens the door to Data Access Objects as well as the automation of applications like Excel and Word. This means you can keep a key field in each contact record that matches the key field in an external database (which might be a desktop database like Access) or an SQL server, and link to this external data as needed (*see "An Outlook example, page 300*). Then you can use Outlook for its address book and calendar features but keep the mission-critical data in a fully-fledged database management system where it belongs.

Embark on Outlook development and you hit the bleeding edge of Office 97. Documentation is scant, some procedures are hopelessly counter-intuitive, the script editor is primitive, and late binding of COM objects means performance is poor compared to real Visual Basic. It is such a

p300 ➢

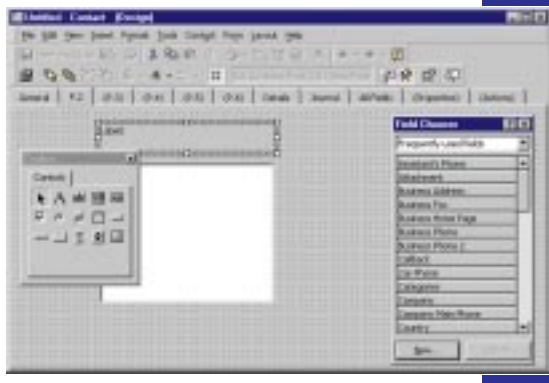


Fig 2 Create a new contact and then choose Design Outlook Form to open the form designer

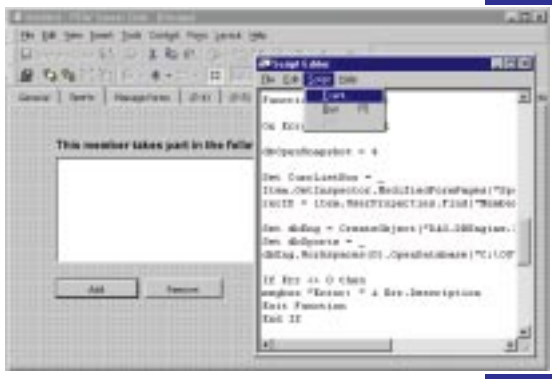


Fig 3 Choose View Code to open the Script Editor. The cunningly-titled Run option does not run the code but merely checks the syntax

Fig 4 When the form opens, Outlook looks up the list of sports from an external database

Creating and saving Outlook forms

Outlook form-handling is perverse. For a start, you cannot just get on and design a new form as you would in Visual Basic. The best way to design a form is to pretend you want to create a new item in the current folder. That opens a blank, default form. Then you can choose Design Outlook Form from the Tools menu. Note that some forms, like the Contacts form, cannot be completely customised. Some parts are read-only. There are plenty of spare tabs, though, which you can use as you want.

A key step is to specify the default form for posting in this dialog

It is when you have designed your form and wrestled with the wretched script editor that the real fun begins. It is no use just saving the form, since all that does is to create one new record with a customised form. You must choose Publish Form As… from the File menu. You can publish to several locations, depending on how widely the form should be available. If you do not have Exchange server, or are working on a test form, the best choice is either your personal forms library or in the specific folder where it is needed. If you choose the latter option, it gets its own entry on the Compose menu.

Next, you have to tell Outlook to use your new form as the default for new items in this folder. To do this, right-click the folder name in the folder list and choose Properties. On the General tab is an option: "When posting to this folder, use…"; here you can specify the new custom form. Now your custom form is used by default for new entries. But it is not over yet. If you try opening any existing items in the folder, you find the old form still being used. The solution is to run a short VBScript routine. Each item in Outlook has a MessageClass property, a concept alien to VB developers but familiar to experts in Microsoft's Mail API, or MAPI. This property determines the form used to view the item.

This is the code to change it:

```
sub UpdateClass

Set currFolder = Application.ActiveExplorer.CurrentFolder
numItems = currFolder.Items.Count

For countvar = 1 to numItems
Set currItem = currFolder.Items(CInt(countvar))
currItem.MessageClass = "IPM.Contact.PCW Sports Club"
currItem.Save
Next

end sub
```

Fig 5: Accessing additional data from within Outlook

```
Function Item_Open()
On Error Resume Next

dbOpenSnapshot = 4

Set CurrListBox = _
Item.GetInspector.ModifiedFormPages("Sports").LstSports
recID = item.UserProperties.Find("MemberID").Value
Set dbEng = CreateObject("DAO.DBEngine.35")
Set dbSports = _
dbEng.Workspaces(0).OpenDatabase("C:\OUTAPPS\SPORTS.MDB")

If Err <> 0 then
msgbox "Error: " & Err.Description
Exit Function
End If

sql = "Select * from sports, sportlink where sports.ID = sportlink.sportID "
sql = sql & "and sportlink.MemberID = " & recID

Set snSports = dbSports.OpenRecordset(sql, dbOpenSnapshot)

If not (snSports.eof and snSports.bof) then
snSports.movelast
snCount = snSports.Recordcount

snSports.MoveFirst

For countvar = 1 to snCount

currListBox.Additem(snSports.Fields("SPORT"))
snSports.MoveNext

Next

End if

snSports.Close
dbSports.Close

End Function
```

nice component in other ways, though, that it is worth persevering. Expect it to get easier in the next version.

An Outlook example

The following example uses the same Sports Club database as the recent *Hands On Visual Basic Workshop* (*PCW* February-May issues).

It is a relational database with three tables. The Members table includes name and address details, and is easily imported into a new Outlook contact folder. When mapping the fields, it is important to include the key field which uniquely identifies each record. Outlook will not let you map an imported field directly to a custom field, so the solution is use a spare built-in field. For a tidy result, you can then add a custom field to hold the ID number and write some code to copy it across. In the example, this field is called MemberID.

Note that not all the data is imported, but only the Members table. In particular, the information about which sports each member enjoys is not available. The trick now is to access this additional data from within Outlook, without actually importing a copy of the tables.

Here's how to do it (also, see Figs 2-4, page 299):

1. Open the contacts folder and choose New Contact from the Contact menu. This opens the built-in Contacts form.
2. From the Tools menu choose Design Outlook Form. This opens the form in design mode with six spare tabs available. Click the second tab, rename it Sports, and add a list box control. Name the list box lstSports. You can add labels and other decoration as desired.
3. On the Form menu, choose View Code. This opens the Script editor. From the Script menu, choose Event and then add the Open event. Fig 5 is the code. Note that you should replace the database filename with the actual location of the data on your system if you use this example. Note also that you can use the Run option in the Script editor to find syntax errors before you save the code. The Run option does not actually run the code as that would be far too easy. Since the script editor has no syntax highlighting and VB Script has no debugger, it can pay to enter and check chunks of code in Visual Basic or Visual Basic for Applications, beforehand.
4. Publish the form and make it the view form for contacts in this folder. This step is so far from being intuitive that I have devoted a separate panel to it (*page 299*).

Now, when you open a contact in this folder, Outlook looks up the list of sports from the original database. Of course, a real-world system could look up a far greater range of information. The important thing is to realise that this sort of link is possible.

PCW Contacts

Tim Anderson welcomes your Visual Programming tips and queries. He can be contacted at the usual *PCW* address or at visual@pcw.co.uk.

The microsoft.public.outlook97 newsgroup is a valuable source of help for Outlook development, as is www.Microsoft.com. Another useful site is www.Outlook.Useast.com.

Sax appeal

Sax Webster is a browser builder that is just the last word in web applications. Tim Anderson models it here for you, taking care not to neglect his widgets and tools while he's at it.

Forget laptops and mobile phones. The fashion accessory of the moment must be the personal web site. Web sites are no use unless they are visited, so why not build point-and-click access into the applications you distribute? You can do this by calling an external application like Netscape or Intenet Explorer, but Sax Software lets you go one better by building a customised browser right into the application.

The Webster control is a 32-bit browser OCX that drops directly into any compatible development tool, such as Visual Basic 4.0 or Visual C++ 4.0. With the rampant growth of the internet and increasing corporate usage of intranet networks, Sax Webster has turned up at just the right moment. For example, online help might now mean dynamic information on a web site, rather than the static file shipped with an application. Another option is to direct the hapless user to a site offering further products and services. HTML pages can be loaded from disk as well as from the internet, so you could also use Webster as a multimedia browser.

Fig 1 All done with Webster: VB 4.0 visits the *PCW* home page

Listing 1: Intercepting the mailto command

```
Private Sub Webster1_DoClickURL(SelectedURL As String, Cancel As
Boolean)
If Left$(LCase$(SelectedURL), 7) = "mailto:" Then
' run MS Exchange, using file association
ShellExecute 0, "open", SelectedURL, "", "", 0
SelectedURL = ""
' stop Webster attempting to act on this command
Cancel = True
End If
End Sub
```

Sax Webster is a complete application wrapped in a control. You can create a browser simply by dropping the Webster control onto a form in VB or Delphi. It claims to support HTML version 3.0, but Sax adds that, "because 3.0 is not yet defined as a standard, it may differ from what Netscape or some other 3.0 browser supports." Here is the problem with Webster and ultimately with the web itself: lack of tightly defined standards, resulting in compatibility problems. It may not matter too much, since it would be foolish to use a Webster application as a replacement for Netscape or Internet Explorer. Webster makes better sense as a tool for accessing specific web sites that are linked to the container application, so you can ensure the compatibility of those particular pages. Some problems can also be overcome by writing code to intercept Webster events. For example, Webster does not support the **mailto** command that HTML uses to initiate an email message. The VB 4.0 code in Listing 1 will intercept mailto and call whatever application is associated with that command in the Windows 95 registry.

Another useful feature is the **GetContent** method, which lets you read all or part of an HTML page into a variable. Initially only

Listing 2: Screensaver application

This application, which toggles the screensaver on and off, needs a VB project with a form, a button and a code module. Note that to work in Windows 3.1, the declarations will need to be adapted.
Code for the form:

```
Private Sub Form_Load()
bOldActive = isActive()
If bOldActive = True Then
Command1.Caption = "Disable screen saver"
Else
Command1.Caption = "Enable screen saver"
End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
SetActive (bOldActive)
End Sub

Private Sub Command1_Click()
 If isActive() = True Then
 SetActive (False)
 Command1.Caption = "Enable screen saver"
 Else
 SetActive (True)
 Command1.Caption = "Disable screen saver"
 End If
End Sub (continues page 285)
```

available as a 32-bit OCX, Sax has now released a 16-bit OCX as well, but nothing yet for VB 3.0 or Delphi 1.0 diehards.

Widgets for your data

Sheridan's Data Widgets has long been one of the most popular Visual Basic add-ons, particularly since the VB 3.0 grid is so poor. The data-bound controls in VB 4.0 are better, but still leave room for third-party enhancements. Version 2.0 brings the expected conversion to 16- and 32-bit OCX format, but with enhancements. Sheridan has taken the opportunity to restructure the data widgets using objects and collections, bringing it into line with other programmable OLE objects. This makes for more logical code and increases the programmer's control, the disadvantage being that code which worked with Data Widgets 1.0 will have to be extensively rewritten. For example, to put a button in a DataGrid cell in version 1.0 used a ColBtn property:

```
SSDbGrid1.ColBtn(2) = True
```

which in version 2.0 becomes:

```
SSDbGrid1.Columns(2).Style = 1
' edit button.
```

The actual Data Widgets controls are the same six as before: Data Grid, Data Combo, Data Dropdown, Data OptionSet, Data Command and the Enhanced Data Control. All are useful but the Data Grid is the reason people buy this package. Its neatest trick is to link with a Data DropDown so that users can click on a grid cell and select values from a dropdown list bound to a field in another table (Fig 3).

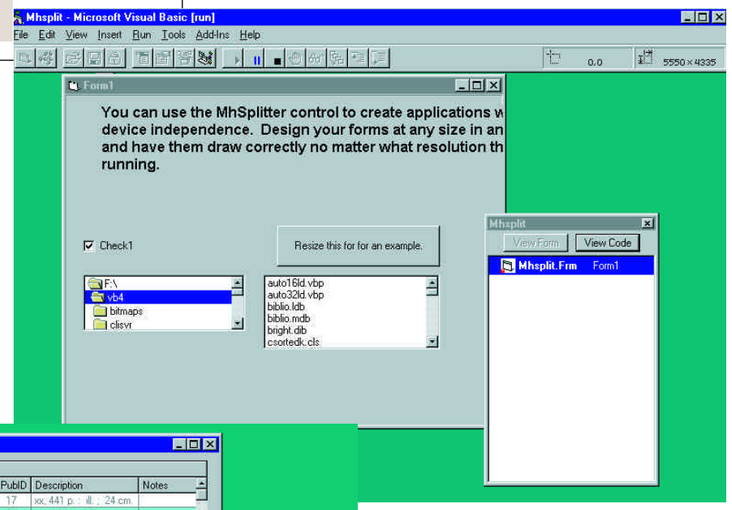


Fig 2 (above) The MhSplitter control from OLE Tools attempting resolution independence. Unfortunately, this text box does not always get resized correctly…

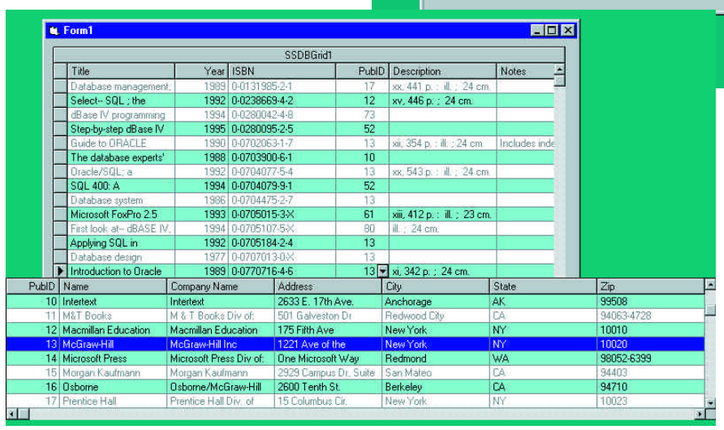


Fig 3 (left) Using a data grid and a data dropdown. Clicking the PubID column drops down the publisher table, so you can see the full details when choosing the ID

p285 ➢

Do you need Data Widgets? It depends entirely on how you prefer to program. If you make extensive use of bound controls, this bundle is all-but indispensable, particularly if a data grid is a key part of the user interface. The data control in VB 4.0 is not compromised in the same way as VB 3.0's effort, so this is a perfectly sound approach. The cautionary note is that large OCX controls like these cause substantially slower loading of your VB application, and that grids are often not the best way to present data to the user. Finally, the Data Grid also works well as an unbound virtual list control, a further enticement which may sway doubters.

OLE tools

Microhelp's OLE tools may have up-to-date OCX technology, yet this package conveys a dated impression. The main reason is that apart from their OCX conversion, many of the controls are little changed from earlier versions, right down to their description in the manual and the clunky example applications. OLE tools also slipped up during review when one of the genuinely new items, MhSubClass, failed to deliver. This is a message-trapping control that can catch Windows API messages and either kill them, or respond with a custom event and then pass them on. MhSubClass is fine for some purposes, for example if you want to inspect WM_MENUSELECT messages in order to provide a help text as the mouse runs down a menu. But a common requirement is to trap a message and then write code to determine whether to kill it or pass it on. MhSubClass cannot do this, since the fate of the message has to be determined before the VB event is triggered. Rivals such as the MessageBlaster OCX have no such handicap.

Never mind the quality. With 54 separate controls, the bundle still rates as good value. MhCalendar is a data-aware calendar control. MhSplitter allows you to build resolution-independence into interfaces by automatically resizing controls within the container, albeit rather slowly (Fig 2). MhRealInput is a text box that improves on VB's masked edit control for working with real or currency values. And so it goes on, providing something of value for most VB projects.

Microhelp supplies two versions of these tools. OLE tools has 16- and 32-bit OCXs, while VB tools stays with the old VBX format. There are differences between the two. For example, the inadequate MhSubClass is OCX-only, while the clever MhOutOfBounds universal data binding control is VBX-only. Finally, VB tools used to come with a version of Farpoint's Grid control, but that has now been dropped.

Listing 2 (continued from page 283)

Code for the module:

```
Option Explicit
Global bOldActive As Boolean
Declare Function SystemParametersInfo Lib "user32" Alias
"SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As
Long, lpvParam As Long, ByVal fuWinIni As Long) As Long
Public Const SPI_GETSCREENSAVEACTIVE = 16
Public Const SPI_SETSCREENSAVEACTIVE = 17

Function isActive() As Boolean

Dim lRetVal As Long
Dim pvParam As Long

lRetVal = SystemParametersInfo(SPI_GETSCREENS AVEACTIVE, 0,
pvParam, 0)

If lRetVal = False Then
MsgBox "Call to SystemParametersInfo failed"
isActive = False
Exit Function
End If

If pvParam = False Then
isActive = False
Else
isActive = True
End If

End Function

Sub SetActive(bActive As Boolean)
Dim lRetVal As Long
Dim pvParam As Long

lRetVal =
SystemParametersInfo(SPI_SETSCREENS AVEACTIVE, bActive,
ByVal pvParam, 0)

If lRetVal = False Then
MsgBox "Call to SystemParametersInfo failed"
End If

End Sub
```

Hacking the system in Windows 95

Mark Horton writes: "*I've just bought a new system with Windows 95 and VB 4.0. My computer has a WIn/TV card, and I wanted to write a program that would turn the screensaver off and on without having to go into the display properties tab. How or where can I find out about the API calls necessary to change the screensaver settings? Is there a book on the market which describes all the Win32 (and/or Win16) API calls?*"

Windows 3.1 introduced a handy function called SystemParametersInfo.

p286 ➢

This reads or sets numerous system parameters including the screensaver settings. Listing 2 *(pp283/285)* shows a small VB application for Windows 95 which toggles the screensaver on and off. The two key functions, IsActive and SetActive, work by calling SystemParametersInfo. The application checks the current state of the screen-saver on loading, so that it can be restored on exit.

Another possibility is for your application to disable the screensaver whenever it has the focus. Windows activates the screensaver by sending a WM_SYSCOMMAND message with wParam set to SC_SCREENSAVE. By intercepting and killing this message, you prevent the screensaver from kicking in. Delphi programmers can trap messages easily, but VB users will need an add-on like the MessageBlaster OCX.

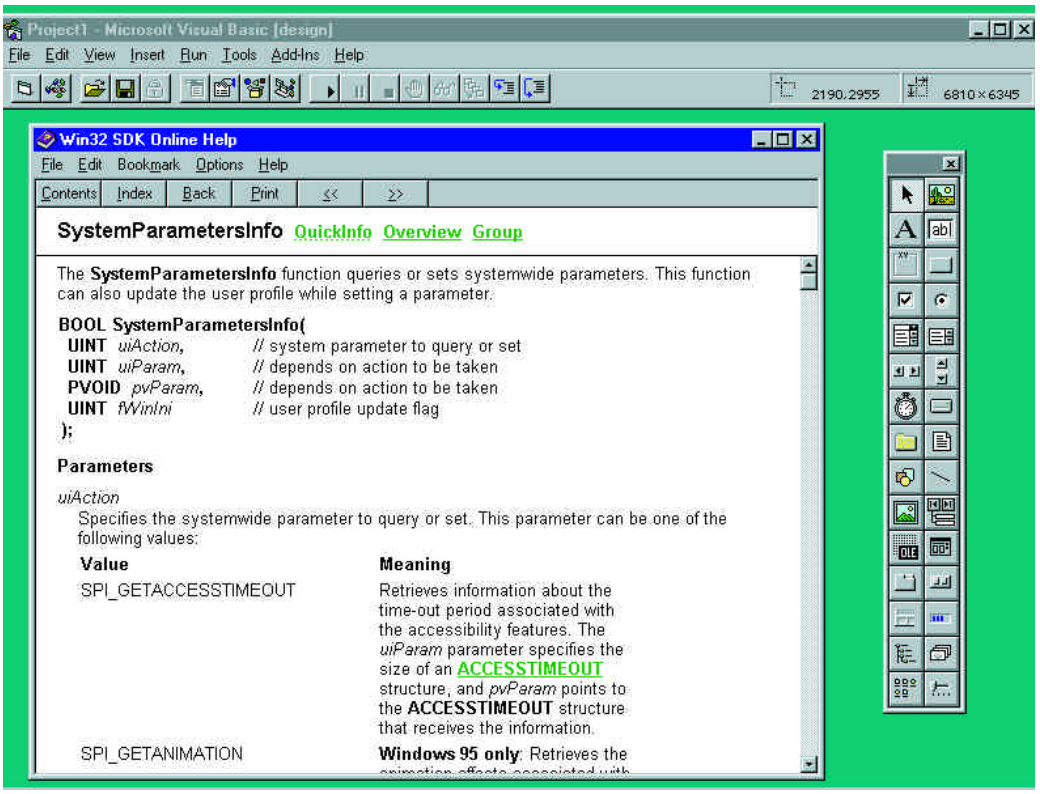


Fig 4 Although aimed at C/C++ developers, the Win32 SDK is an essential reference for Visual Basic developers. So why is this help file not supplied with Visual Basic 4.0?

Many problems like this can only be solved using the Windows API. That in turn means having a good API reference, and the starting point is the Windows SDK help file (Fig 4) called WIN31WH.HLP for Windows 3.1 and WIN32.HLP for 32-bit Windows. Surprisingly, Visual Basic 4.0 comes with declarations for the 32-bit API but not the 20Mb help file. An alternative is Daniel Appleman's book, *VB Programmer's Guide to the Windows API*, which provides what is needed for Windows 3.1 and is to be updated for Win32.

Listing 3: ParamCount

```
procedure TForm1.Button1Click(Sender: TObject);
var
i: integer;

begin
for i := 0 to ParamCount do
   MessageDlg(ParamStr(i), mtInformation,
      [mbOk], 0);
end;
```

Listing 4: BeginUpdate

```
procedure TForm1.Button2Click(Sender: TObject);

begin
listbox1.items.beginupdate;
listbox1.items.add('One item');
listbox1.items.add('another item');
listbox1.items.endupdate;
end;
```

Tips for Visual Programming

- Speed VBs load time and slim your applications by stripping down AUTOLOAD.MAK (VB3) or AUTO32LD.VBP (VB4) to include only controls and references essential to every project.
- Avoid Dim iA, iB as Integer. This code declares iA as a variant. Instead, use Dim 1A as Integer, IB as Integer.
- In VB4, disable Compile on Demand (in Tools - Options - Advanced) to have the compiler check for syntax errors before a project runs.
- Your Delphi application can easily check for command-line parameters. ParamCount returns the number of parameters; ParamStr(0) returns the path and filename of the application, and ParamStr(n) returns the nth parameter up to ParamCount. (Listing 3)
- If you are adding lines to a string control like a listbox or memo, or an outline component, use BeginUpdate to increase performance by preventing screen updates. (Listing 4)

PCW Contacts

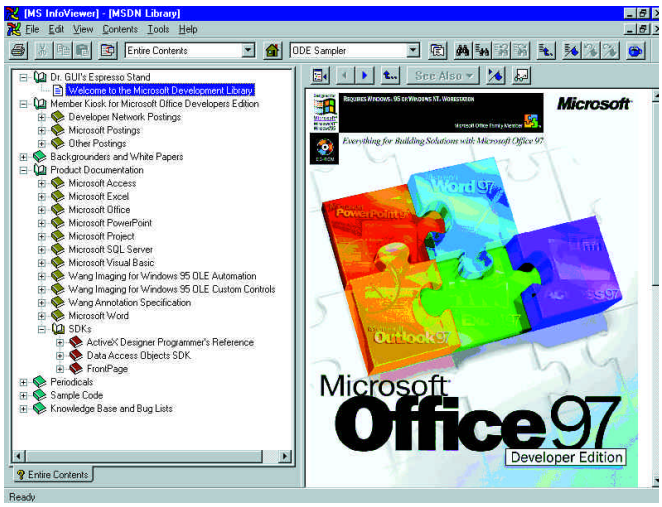
Tim Anderson eagerly awaits your comments, queries and tips, either at the usual *PCW* address or by email at visual@pcw.co.uk.
Visual Basic Programmer's Guide to the Windows API by Daniel Appleman (Ziff-Davis Press, £33.02)
Computer Manuals 0121 706 6000
Sax Webster £110 (plus VAT)
Data Widgets 2.0 is £99 (plus VAT)
OLE Tools is 149.00 plus VAT and **VB Tools** £99 (plus VAT) from **Contemporary Software** 01727 811999

Open and shut case

Office 97… just more of the same and not worth the upgrade? To casual users, maybe; but for developers, it offers a more open programming environment. Tim Anderson explains.

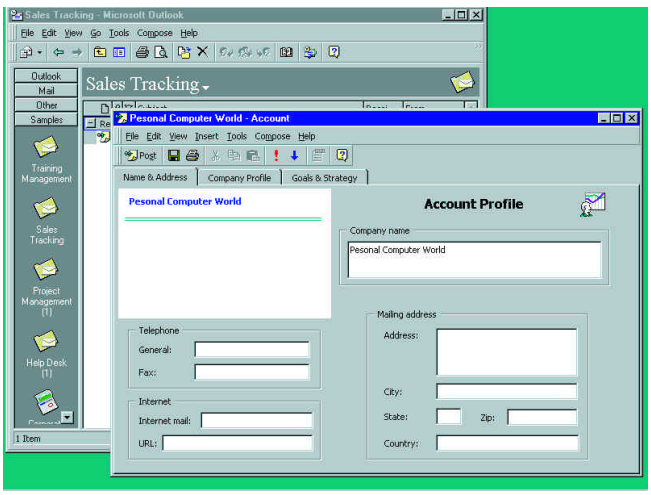
I have heard muttering to the effect that Office 97 is not much different from Office 95, or even Office 4.x. From some perspectives, that is correct. Word looks similar, Excel looks similar and all the casual user will notice at first is the Office Assistant (fantastic or horrific, according to taste) and a new, flatter, look to the toolbars.

But developers should welcome Office 97 with open arms. It is even worth explaining to the users why they really should upgrade, even if the animated Clippit is not their cup of paperclips. The reason is Visual Basic for Applications combined with the updated Office object model, all of which is exposed for programming. In most cases, equipping an Office user with suitable templates and macros soon pays for itself in increased productivity.



Left The Office 97 Developer edition is great if you need Access runtime, but otherwise it is not essential

Below, left You can develop applications in Microsoft Outlook by creating custom forms driven by VB Script. Nearly wonderful, but not quite there yet



To do these wonderful things, you need appropriate resources. This might or might not include the Office 97 Developer Edition, Microsoft's one-stop solution. What you get is the Office Professional CD, plus a further CD of developer tools. There are also two books: the *Office 97 Visual Basic Programmer's Guide* and *Building Applications with Access 97*, along with a booklet of Object Model charts.

Before getting too excited, though, it is worth recalling that the original Office Development Kit CD used to be more or less given away by Microsoft, presumably on the grounds that it pays to have people develop Office solutions, since in order to deploy them a copy of Office must be purchased for each installation.

Another noteworthy detail is that the books in the Developer Edition are available separately from Microsoft Press. A third observation is that not all the documentation you might need is actually included, neither on paper nor online. A notable example is *Building Microsoft Outlook 97 Applications*, which would be particularly valuable as Outlook is brand new. Another noteworthy example is the Office 97 Resource Kit: the version on the CD is for Office 95, or at least it is in my US shrinkwrap copy. The resource kit is aimed at network administrators but contains useful information for developers as well.

The most essential developer reference, the VBA reference for each application, is actually on the Office 97 Professional CD so it is questionable whether the Developer Edition is worth having. Most of the material can also be found on the Microsoft web site, often in updated form. There is only one convincing reason for buying this product: to obtain the runtime version of Access 97. This gives you a licence to deploy Access applications royalty-free, and could soon pay for itself. If you don't need it, just subscribe to MSDN (or buy a library

p302 ➢

Fig 1 Using the clipboard

```
Dim cr As String
cr = Chr$(13) & Chr$(10)
RichTextBox1.Text = “This is a picture” + cr
RichTextBox1.SelStart = Len(RichTextBox1.Text)
RichTextBox1.SetFocus
Clipboard.Clear
Clipboard.SetData Image1.Picture, vbCFBitmap
SendKeys “^v”
```

CD from time to time) and buy the books you really need from a bookshop.

Outlook: nearly great

Is it a Personal Information Manager, or an email client, or maybe groupware? Outlook is ambitious, and nearly the foundation of a complete Office solution. For example, it should be possible, with a bit of customisation, to right-click a contact name and open up a customer’s order history or an index of previous correspondence.

There are two snags, though. One is that Outlook has VB Script but not yet Visual Basic for Applications. The second is that you need Exchange Server to do anything serious with Outlook over a network, like sharing an address book or viewing other people’s calendars. In fact, Outlook without Exchange Server is less capable than the old Schedule, a fact which has not gone down well with small businesses running peer-to-peer networks. Exchange Server needs Windows NT, is priced for the Enterprise market and needs client licences, too, which makes Outlook far less attractive. Incidentally, if you decide to get going with Outlook development, a trip to Microsoft’s web site is essential. Documentation and numerous sample applications are available for free download.

Sheridan’s Active Threed

Sheridan products now come on a Toolkit CD containing all the company’s developer tools. You can install demonstration versions of any tool, or full versions where you have the right key code. For instance, if you purchase Active Threed you get the code for this product along with the CD. There is no manual, the lame excuse being that Sheridan wanted to check the printed manual against the release code. A voucher lets you obtain it at nominal cost. But the manual aside, the all-in-one CD is a great idea. Another plus is that the ActiveX controls come digitally signed with CAB versions included for web distribution.

Active Threed offers seven controls which are intended as plug-in replacements for the standard Windows items like command buttons and check boxes, but with extra features. These include marquee captions which blink, scroll, slide and bounce, plus animated pictures created with a sequence of bitmaps. There is also a splitter control which lets you create windows with resizable panes. The controls are 32-bit only; not even Visual Basic 4.0 16-bit is supported. Packages like this cause me to hesitate since many VB applications are slow enough without the additional weight of controls which aren’t strictly necessary.

Two things make ActiveThreed worth a second look, though. Firstly, the SSplitter control is well implemented and provides a feature which has become something of a Windows standard. Secondly, the SSRibbon control allows you to create toolbar icons with an active border, as seen in Office 97 and Internet Explorer 3.0. I was also glad to find that Delphi samples had been included.

Rich Text in Delphi and VB

Dr Francis Burton asks: “*I want to be able to alternate graphics and text in a scrollable window, with the ability to cut and paste text (and possibly bitmaps/metafiles, too). New text and graphics are appended to the end of the window. Do you know of any controls, either Visual Basic or Borland Delphi, which implement scrollable graphics/text windows?*”

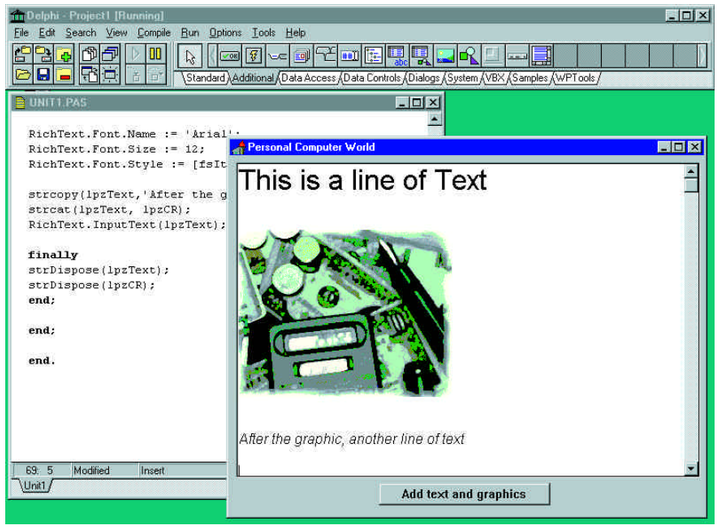
If you are working in Windows 95 or NT, the standard rich text control can display formatted text and graphics. It is easy to miss this functionality in Visual Basic, since there is no InsertPicture method. You can insert OLE objects, though, using the OLEObjects collection. For example, this code inserts a line of text and a picture:

```
RichTextBox1.Text = “This is a
picture”
RichTextBox1.OLEObjects.Add , ,
“c:\test.bmp”
```

Unfortunately, this can have unpredictable results depending on how OLE file associations are set up in the registry. There is also an overhead involved with OLE which makes the rich text box update rather slowly when an object is inserted. A safer approach would be to use the clipboard. The example in Fig 1 and the Delphi one (Fig 2) assume you have placed the picture you want to insert into an invisible image control on a form. The final

Fig 2 Using WPTools

```
var
lpzCR: pchar;
lpzText: pchar;

begin
lpzText := stralloc(256);
lpzCR := stralloc(3);

try
strcopy(lpzCR,chr(13));
strcat(lpzCR, chr(10));

RichText.clear;
RichText.Font.Name := ‘Arial’;
RichText.Font.Size := 24;

strcopy(lpzText,’This is a line of Text’);
strcat(lpzText, lpzCR);

RichText.InputText(lpzText);

RichText.PicInsert( image1.picture, 0,0 );
RichText.InputText(lpzCR);
RichText.Font.Name := ‘Arial’;
RichText.Font.Size := 12;
RichText.Font.Style := [fsItalic];

strcopy(lpzText,’After the graphic,
another line of text’);
strcat(lpzText, lpzCR);
RichText.InputText(lpzText);

finally
strDispose(lpzText);
strDispose(lpzCR);
end;
```

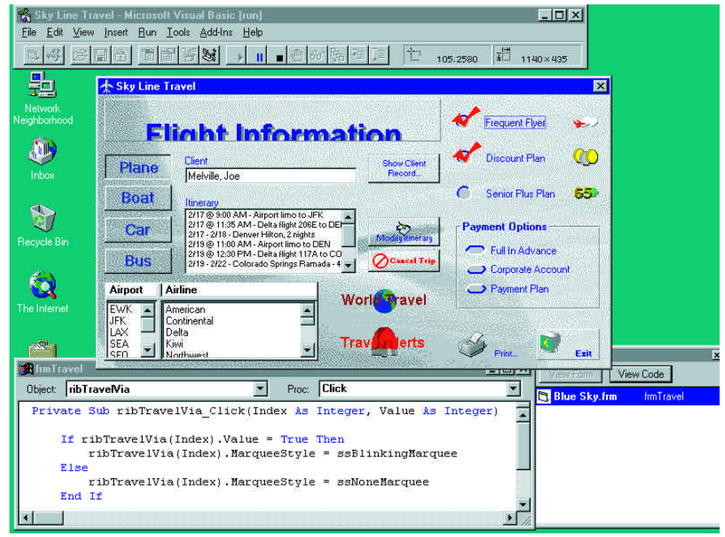
Even Delphi 1.0 is able to display text and graphics within a scrolling document using WPTools (illustrated here), or a component such as Visual Writer

SendKeys statement simulates the CTRL-V keypress which pastes from the clipboard at the insertion point.

For some reason, Delphi’s equivalent rich text control does not support graphics. It is odd, since both use the same underlying common control, and it is likely that careful investigation of the Visual Component Library would reveal a way to overcome the problem by creating a new component which exposes more of the features in the rich text control. Or, you could use VisualWriter, an OCX and VBX control which does support graphics. Better still, use a native Delphi component like WPTools (it’s shareware but works well). The WPRichText control, part of WPTools, has a PicInsert method which lets you insert a picture. It also support fonts and styles so you could implement scrollable text and graphics as required. The main snags with WPTools are its uneven documentation, and extensive use of pointers which can be error-prone. Fig 2 shows example code using WPTools.

String along with SQL

Michael O’Reilly writes: “*I have been following your excellent VB tutorial, and while using some of the code given in the February issue article I encountered a problem I can`t solve. In the example given, you take a string from a text box and use it in an SQL query. How do you do the same with a number from a text box?*”



A printed picture does no justice to this Active Threed form, which is crawling with animation. Note the split window at bottom left — a genuinely useful feature

When you create an SQL string for querying a database, you use a different technique according to whether the field is character or numeric. If it is the former, the value must be in single quotation marks, as in:

```
sSql = “select * from
members where
members.surname = ‘“ &
mySurname & “‘“
```

Admittedly this looks ugly, but it works well. If the field is numeric, then the single quotation marks must not be used. All you need to do is convert the numeric value to a string like this:

```
sSql = “select * from members where
members.surname = “ &
str$(myID)
```

The following question comes from Andrew Shaw: “*I need a lot more input boxes than the PCW Sports Club uses and want to implement some kind of counter/loop to run through the DisplayPerson code. I want to avoid:*

```
txtForename = CurrPerson.Forename
txtSurname = CurrPerson.Surname
```

“*I tried an array of text boxes and a laborious trawl through the manual, with no success. I want to try something like:*

```
txtInput(Counter) =
CurrPerson.Counter
```

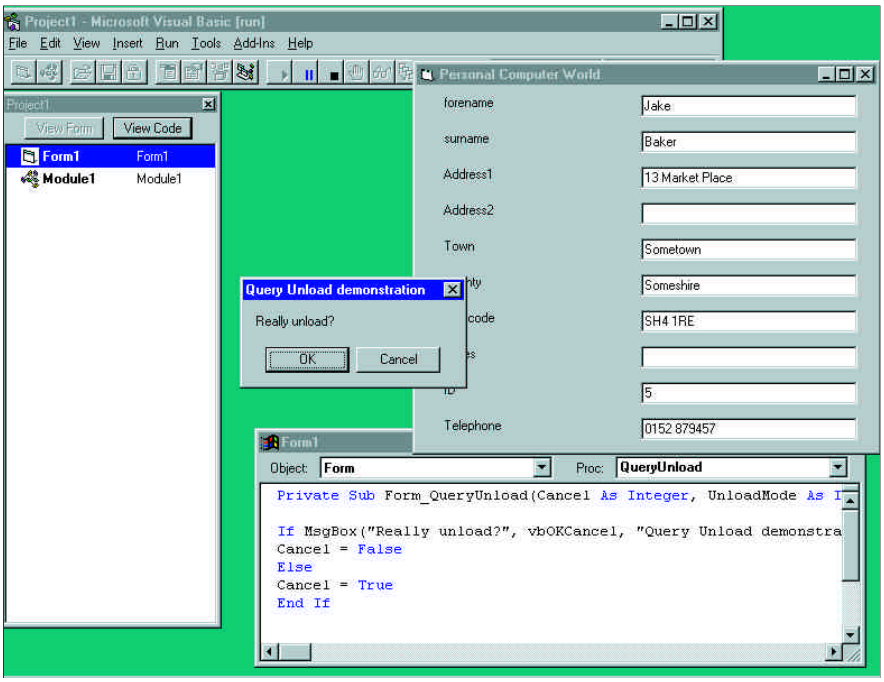
“*What should the CurrPerson.Counter part look like?*”

Andrew’s idea is to write a loop that iterates through all the fields of a particular record, filling text boxes with the values along the way. This can be done as follows:

```
Dim iCountvar As Integer
For iCountvar = 0 To
 (ds.Fields.Count - 1)
Label1(iCountvar).Caption =
 ds.Fields(iCountvar).Name
Text1(iCountvar).Text = “” &
 ds.Fields(iCountvar).Value
Next
```

The trick is to get at the Fields collection of a Recordset object. You could make the routine even more flexible by creating the necessary labels and text boxes at runtime.

p305 ➢



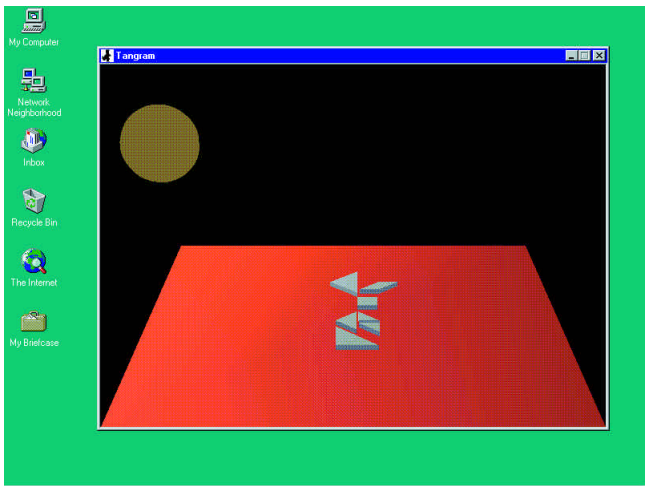


Fig 3 (above) Using the QueryUnload event to confirm a close decision

Left (see "Inside Com") This application from Inside COM demonstrates agregation, containment, and interchangeable components. The Tangram pieces look like a rabbit… allegedly

The main value of a routine like this is in applications where the number and type of fields in the recordset may vary at runtime. For example, you could let the user choose which fields they wanted to view, build an SQL string to return just those fields, and display them using the procedure described.

Where's the tab strip?

Phil Richard asks: "*In a recent column, you mentioned a tab strip option in VB4 which I would like to use; either TabStrip or SSTab. Neither seem to be included in my installation of Standard Edition VB4, which I purchased as an upgrade. I have, however, found TABCTL32.OCX from the Sheridan web site. Is there a way of registering it, as it appears as not found when trying to load the PCWClub project.*"

Unfortunately Phil is correct, and the Tab custom controls are only present in the Professional edition of Visual Basic. While a lot can be done with the Standard version, it is severely restricted both in its use of the JET database and in the number of custom controls supplied. It is also cheap, and my guess is that Microsoft views it as an introductory, learning product rather than a real development tool.

With TABCTL32.OCX, most third-party OCX vendors allow free distribution of its controls. So in order to make some sales, use of an OCX in developing an application is allowed only if you have purchased the control. When you do buy it, you get either a .LIC file or special registry entries that allow you to use it for development.

How do I cancel?

Ammar EL-Hassan has this query: "*I am trying to add a facility to enable the user of my VB application to CANCEL the operation which Closes his form. The user clicks the control box in the top left corner of the form. They then click the Close option, which unloads the form (application dead!). How can I interrupt this to enable the user to cancel after selecting Close?*"

VB forms have a QueryUnload event for this purpose (Fig 3). Use code like this:

```
Private Sub Form_QueryUnload(Cancel
 As Integer, UnloadMode As Integer)
If MsgBox("Really unload",
vbOKCancel) = vbOK Then
Cancel = False
Else
Cancel = True
End If          End Sub
```

You can even discover why the form is trying to close, by inspecting the UnloadMode parameter. If it is vbAppWindows, then the user is trying to close down Windows.

Inside COM by Dale Rogerson

Inside COM is a book that takes you step by step through the mysteries of COM interfaces, reference counting, globally unique identifiers, containment, aggregation and automation. All the examples are in C++ but the author has avoided Windows-specific code where possible. The strength of the book is that it is about COM rather than OLE or ActiveX technologies which are based on COM, so it does a good job of explaining what COM is and how it works. Of course, the impressive thing about tools like Visual Basic and Delphi is that you can use COM without needing to understand much about it. When it comes to advanced development or troubleshooting, though, a book like this provides an invaluable background.

PCW Contacts

Tim Anderson welcomes your Visual Programming comments, queries and tips. Contact him at the usual *PCW* address or email visual@pcw.vnu.co.uk.

Office Developer Edition is £639 (ex VAT). Upgrades from Office 97 Professional are £215 (ex VAT). Contact **Microsoft** 0345 002000
Office 97 Visual Basic Programmer's Guide available separately at £32.49 from **Computer Manuals** 0121 706 6000
Sheridan Active Threed £99 (ex VAT) from **Contemporary Software** 01344 873434
Inside COM (Microsoft Press) is £32.99 from **Computer Manuals** 0121 706 6000.
Visual Writer is £195 (ex VAT) from **Visual Components** 01892 834343
WP Tools is shareware. Contact Julian Ziersch 100744.2101@compuserve.com

Clean-up campaign

Tim Anderson wrestles with the registry in an attempt to unscramble his settings, tries to get Access from Delphi, and plays Sherlock Holmes to detect which applications he has running.

Imagine you have paid a four-figure sum for a top-of-the-range client-server development system. One day you open up the development environment and the splash screen declares it to be the entry-level hobbyist version. Next, you open the application you are working on to be informed that you are not licensed to use some of its components. Sighing, you reinstall the product from CD but it does not fix the problem.

Sounds fun? This is exactly what can happen with Visual Basic 4.0. The reason, as you will have guessed, is that both VB itself and the many OCX controls which come with it depend on numerous registry settings. If the registry gets scrambled, this is the kind of thing that can happen.

The good news is that Microsoft's web site has a fix. Article Q149619 is entitled "Visual Basic displays incorrect splash screen", although the splash screen is the least of your problems. It is not such good news though. The official fix goes as follows:

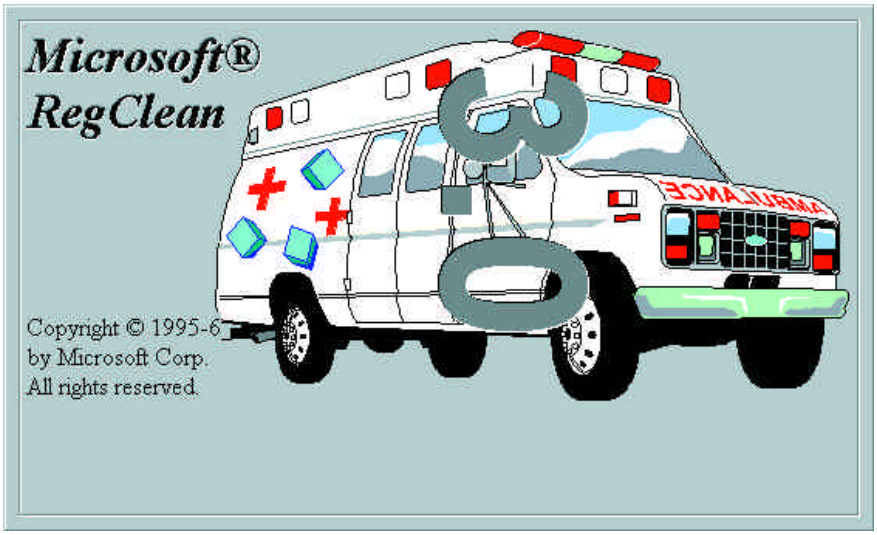
1. Using a registry editor, delete the HKEY_CLASSES_ROOT\LICENSES key.
2. Run Regclean.exe and delete all *.OCX and *.OCA files.
3. Delete OLEPRO32.DLL.
4. Restart Windows and reinstall Visual Basic.

Is this a good fix? Well, it's better than destroying your hard disk with a sledgehammer, but not much. As a developer, you will know that those .OCX and .OCA files represent most of the ActiveX controls on your system. An OCA file, by the way, is an OLE-type library created by VB when you first load an OCX. And ActiveX, says Microsoft, is becoming the foundation of Windows. Then there is the license key to think about, which any number of applications may be using. A clue to the extent of this devastation is given in the note at the end of the fix. "Reinstall third party custom controls," it says, "and any software that may use the registry to store licensing information."

As for Regclean, a utility that comes with VB, I have come to mistrust it deeply. In a misguided moment I ran the latest version 3.0 which you can download from www.microsoft.com. The idea was to fix the annoying messages VB gives you when something is awry in the registry: "Object server not correctly registered". To my great amusement, the end result was worse. Post-Regclean, VB gave me this inspiring piece of technical information 148 times before it would open the Custom Controls dialog. At times like that, you reach for your registry backup with relief.

This problem is not going to go away. An added twist is that the software industry now gives huge distribution to beta versions, via demonstration CDs and over the web. We are all encouraged to spend our time installing trial software, often laden with ActiveX elements, and probably fixed to stop working after a certain date. Frankly, the registry stands no chance of staying clean in these circumstances. Naturally, it is not just developers who install all this stuff, but clients and users as well. Any application that uses standard Microsoft or third-party ActiveX controls or servers may find the ground sweetly removed from under its feet. In the meantime, here are my tips for avoiding registry hell:

1. Check your registry backup procedures.
2. Press Microsoft to come up with proper registry management tools, rather than these draconian "delete everything and reinstall" solutions.
3. Install beta software on a machine



Microsoft's RegClean 3.0: proceed at your own risk

p308 ➢

Borland's Developer Conference CD has some great resources, but why pay when you can visit the web site?

Thanks to the popularity of Microsoft Office Professional and Visual Basic, desktop data is frequently stored in Access MDB files. This creates a problem for other applications which need to get at the data, especially since Microsoft has never documented the structure of an MDB. In any case, the format changes with each new release of Access. Borland's Database Engine can only get at an MDB through ODBC, which is the method Guy has tried. Sadly, the BDE is not at its best with ODBC, and Microsoft's ODBC drivers for Access are nothing special either.

The situation is complicated by the inclusion of ODBC drivers with Microsoft Office, that are designed only to work with Office applications. This might well cause the error Guy is seeing. It is important to get hold of the separate ODBC desktop driver pack, for example from the Microsoft Developer Network CDs, but even then it might not work. It needs the right combination of DLLs, registry entries and even INI files to work as it should, and one or other can easily get corrupted. Sometimes the only solution is to remove dedicated to that purpose. Do not install it on a system used for real work.

4. Persuade your users to adopt the same policy.

5. So you only have one PC? Well, you have been warned.

Mixing Delphi and Access

Guy Cartwright writes: "I'm led to believe that, using Borland's Database Engine, I can access data stored in a Microsoft Access database. I've followed the procedure in a book and created an alias called TstAcess, but I get the message 'Application is not enabled for use with this driver. Alias: TstAccess'. I've trawled the net for an answer but to no avail."

Delphi

Borland's Conference CD

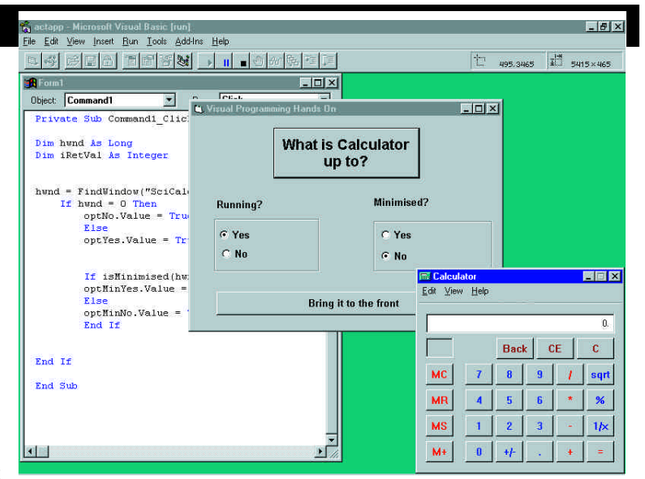
Borland developers who look with envy at the Microsoft Developer Network CDs, stuffed with documentation and tips, will be interested in the recently issued Developer Conference CD. At first glance it looks great, with technical papers and example code covering many real-world problems. The two most prominent products are Delphi and C++ 5.0. The catch is that what you get depends on whether individual speakers at the 1996 Borland conference bothered to send in their notes.

For example, an entry on "Client server development using Delphi and Oracle" leads to a detailed article with source code and a Powerpoint slide show, while another entitled "Rapid application with Delphi 2.0" brings up only a speaker biography. Everything is in HTML and no search program is provided, so you are left to use your own search tools. You also get a collection of patches, technical notes and demonstration versions. Overall there are plenty of good nuggets of information, but it is all rather a mish-mash and mostly available free from Borland's web site. A useful resource, but not for the price Borland is asking.

Fig 1 Routine written from the DAO COM interface

```
var
sSql: string;
dbEngine: Variant;
db: Variant;
snMembers: variant;

begin

sSql := 'Select * from members order by surname;';
dbEngine := CreateOleObject('DAO.DBEngine');
db := dbEngine.OpenDatabase('C:\DATA\SPORTS.MDB');
snMembers := db.OpenRecordSet(sSql, 4);
{4 is dbOpenSnapshot}

If not snMembers.EOF Then
begin
Edit1.text := snMembers.Fields['SURNAME'].Value;
end;

snMembers.close;
db.close;

end;
```


Powers of detection



Using API functions you can find out which other applications are running

Once you get started with Windows programming, you soon find you need to communicate with other applications. At its simplest, for example, you might want to run the Windows calculator from a menu option in a VB application. Easily done with the Shell function but what if the Calculator is already running? In that case, you probably want to bring forward the existing instance rather than starting a new one. Here is how you can find out.

The key to detecting an application is to look for its main window. The API offers functions for listing or searching all the current windows. FindWindow takes two parameters, both null terminated strings. The first is a classname, the second the text of a window title. You can search for one or both and if it finds a matching top-level window, FindWindow returns the handle. For example:

```
hwnd = FindWindow(vbNullString, "Calculator")
```

If it returns 0, then Calculator is not running. Of course FindWindow must be declared, and you can copy the declaration from VB's API viewer.

In the example above, FindWindow searched for the window title. This works fine with Calculator, although you could not be sure which calculator you were getting. It falls down with MDI applications, where a maximised document window adds its title to the main window. You might want to use the classname instead. It is not obvious what the right classname is, but there is another API function, GetClassName, which reveals all. Calculator turns out to have a classname of "SciCalc", while Word is "OpusApp". VB is "ThunderMain", and a VB application, "ThunderForm" or in version 4.0, "ThunderRTForm". Delphi applications get their classname from the name of the main application window, for example "TForm1". So the decision to look for a classname, a window title or both depends on which application you are trying to detect.

If the application is running, the next step is how to bring it forward. One possibility is the API function BringWindowToTop. For example, the following code detects Word and brings it forward if found:

```
hwnd = FindWindow("OpusApp", vbNullString)
    If hwnd <> 0 Then
    BringWindowToTop (hwnd)
   End if
```

The one time this will fail is if Word is running but minimised. A minimised window brought to the top is not much help. Time for another API function or two, in this case GetWindowPlacement and ShowWindow. Using the API viewer, add the declarations for the following:

```
GetWindowPlacement
ShowWindow
Type POINTAPI
Type RECT
Type WINDOWPLACEMENT
Public Const SW_SHOWMINIMIZED
Public Const SW_RESTORE
```

You can now discover whether a non-VB window is minimised like this:

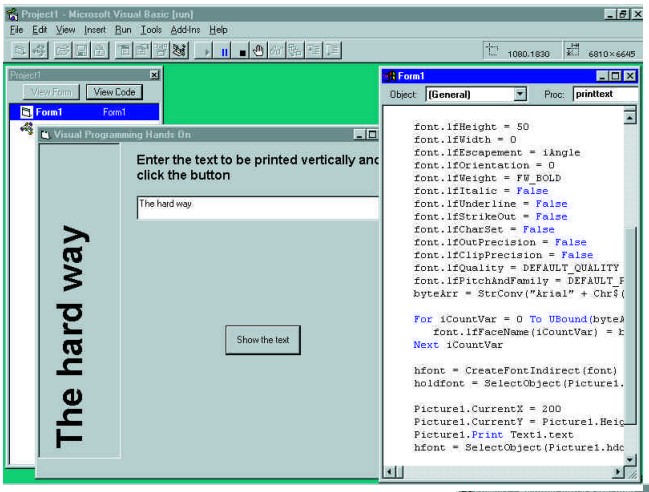
```
Function isMinimised(hwnd) As Boolean

    Dim lpWnd As WINDOWPLACEMENT
    lpWnd.Length = 44 ' 22 in 16-bit Windows
    Call GetWindowPlacement(hwnd, lpWnd)

    If lpWnd.showCmd = SW_SHOWMINIMIZED Then
    isMinimised = True
    Else
    isMinimised = False
    End If

End Function
```

Now the function for bringing Word forward can be modified as follows:

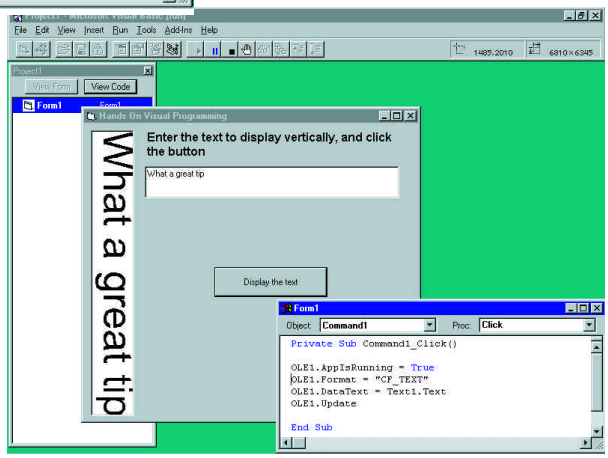
```
hwnd = FindWindow("OpusApp", vbNullString)
  If hwnd <> 0 Then
    If isMinimised(hwnd) Then
    iRetVal = ShowWindow(hwnd, SW_RESTORE)
    Else
    BringWindowToTop (hwnd)
    End if
  End if
```

If you look up GetWindowPlacement and ShowWindow in an API reference, you will find numerous other fields and parameters that give you fine control over the results. One point to notice is that the length field of a WINDOWPLACEMENT type (or structure in C) must be set before it is passed as a parameter in GetWindowPlacement. Unfortunately VB has no SIZEOF function, so you cannot do this neatly. All you need to know for the moment is that in 16-bit Windows the magic number is 22, and in 32-bit Windows it is 44. Occasional inconveniences like this are the price you pay for avoiding the intricacies of C.

p311



Left Vertical text the hard way, setting the font with the Windows API



Below Vertical text the easy way, using the OLE container and a WordArt object

both the ODBC driver and the BDE, weeding out any registry entries as well, and then to reinstall them both. Microsoft Query, which comes with Office, lets you test ODBC data sources by running queries against them.

There is another option if you are running Windows 95 or NT. Microsoft has created a COM interface to the JET database engine under the name Data Access Objects (DAO). It is documented and can be called from Delphi, and you can write routines like in Fig 1 *(page 308)*. For this to work, DAO must be installed on the system, as it will be if you have Microsoft Office 95, for example.

There are several other problems. Microsoft's documentation is aimed at users of Visual Basic or Visual C++, so you have to feel your way to some extent. None of Delphi's data-aware components will work. Finally, you cannot freely distribute the DAO files with a Delphi application. All but the last can be fixed by buying a third-party tool for using DAO with Delphi. Two well-known ones are Titan Access and Opus DirectAccess, while Nortech Software has a third in preparation. One of these is likely to be the smoothest route towards using Delphi with Access MDBs.

Visual Basic

Going vertical

Andy Smith asks: "How can I print vertical text in a Visual Basic application?"

There are a couple of easy solutions, and a better but more difficult one. The easy way is to use a paint program to rotate some text — Windows Paint or the shareware Paintshop Pro will do nicely — and paste it into an image control. You could even have several different messages and load them at runtime. For the best performance, do not load them from disk but use invisible image controls, or the PicClip control, or the Imagelist control.

If you want to be able to specify any text you like at runtime, another possibility is to use the WordArt applet that comes with Microsoft Office or Publisher. Here's how:

1. Pop an OLE container onto a form and set it to contain a new WordArt 2.0 object.
2. Right-click the OLE container and choose Open. In the WordArt dialog, choose the text shape and font required.
3. Use code like this to update the text at runtime:

```
OLE1.AppIsRunning = True
OLE1.Format = "CF_TEXT"
OLE1.DataText = Text1.Text
OLE1.Update
```

The snags with the WordArt approach are firstly that you need the applet installed on the user's system, and secondly a little overhead thanks to OLE. If that rules it out, the heavy coder's method is to call the Windows API. Windows uses a structure called a LOGFONT to define font characteristics, including several properties not exposed by VB's Font properties. One of these is lfEscapement, which specifies the angle of the text. Assuming that the y co-ordinates count from top to bottom, the lfEscapement field specifies the anti-clockwise angle in tenths of a degree. That means you can print diagonal text or even write a routine using a timer that would rotate text around a central point. To set a font using the API, take the following steps:

1. Declare the necessary API types, constants and functions.
2. Define the fields of a LOGFONT variable.
3. Create a logical font by calling Create-FontIndirect. This returns a handle to a font.
4. Select the font into a device context by calling SelectObject. For example, VB Picture Boxes, Forms, and the Printer object all have hdc properties which give you a handle to the device context.
5. Print to the device context using VB's print method or API functions such as TextOut or DrawText.
6. Clean up by unselecting the font and calling Delete-Object with the font handle.

Minimal sample code for drawing vertical text in VB 4.0 is included on the CD. Similar code works in VB 3.0 or 16-bit VB 4.0. It seems complex at first but it is the kind of code you can use again. Then again, alongside the four lines needed to automate WordArt, it does look like an argument for sticking to the easy way.

Cover CD

The MSDN starter edition for Visual Basic is on this month's cover-mounted CD-ROM. It includes 125Mb of searchable information on VB 3.0 and VB 4.0.

PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address or at visual@pcw.vnu.co.uk

Borland Developers Conference CD £59 (plus VAT) from Borland 0800 454065
Delphi 2 Developer's Guide (Pacheco and Teixeira) from SAMS/Borland Press £54.99
Opus DirectAccess £189 (plus VAT) from QBS 0181 956 8000, www.opus.ch
Nortech Software is at www.wizzkids.com
Titan Access 32 is £225 (plus VAT) from QBS 0181 956 8000, www.reggatta.com

A new Deal

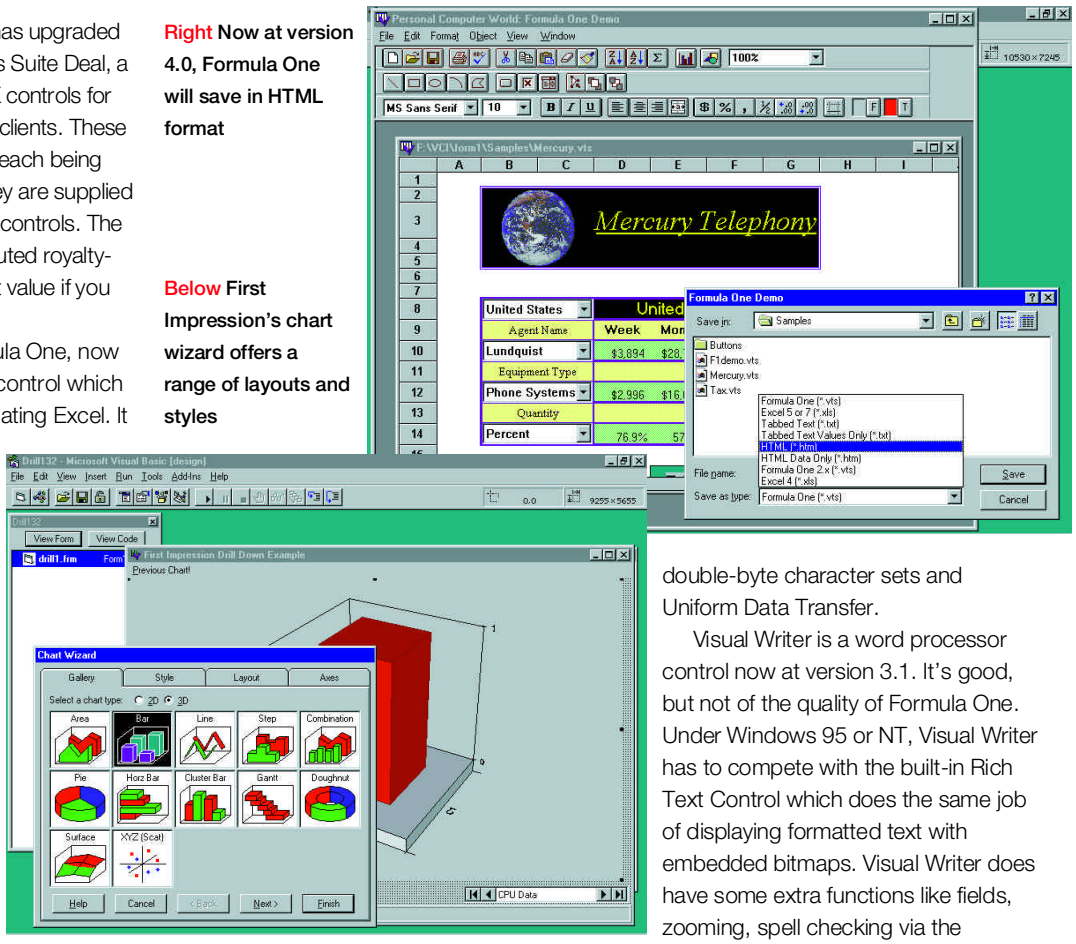
Tim Anderson checks out the new Formula One spreadsheet control in the latest upgrade to the Visual Developers Suite Deal, answers VB and Delphi queries and hides a blinking caret.

Visual Components has upgraded its Visual Developers Suite Deal, a collection of ActiveX controls for Visual Basic and other ActiveX clients. These are heavyweight components, each being almost an application itself. They are supplied as both 16-bit and 32-bit OCX controls. The runtime versions can be distributed royalty-free, making the Suite excellent value if you need this kind of functionality.

Cream of the crop is Formula One, now at version 4.0, a spreadsheet control which does a remarkable job of emulating Excel. It can read and write files in Excel format up to version 7.0, as in Office 95, but there are limitations: Formula One does not understand Excel charts or macros, for example. A large number of worksheet functions are supported, and the ability to move sheets to and from Excel is a valuable asset. Formula One has its own drawing tools and can link with First Impression, the charting control in the Suite Deal, to create charts. You can place buttons, checkboxes and drop-down listboxes on sheets.

Right **Now at version 4.0, Formula One will save in HTML format**

Below **First Impression's chart wizard offers a range of layouts and styles**



New in version 4.0 is support for double-byte character sets, HTML export, and Uniform Data Transfer, an OLE standard which lets you drag and drop data between applications. Formula One is not a data-bound control, but it has built-in ODBC support so you can query an ODBC database and the results appear in a worksheet.

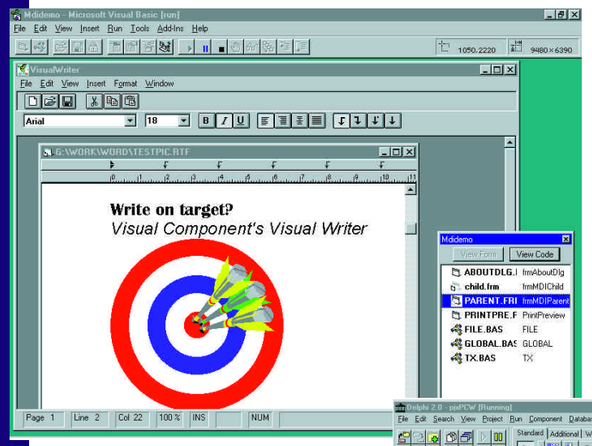
Version 4.0 includes several new ODBC functions. Another nice touch is the workbook designer, a fully-featured spreadsheet application which pops up on demand to enable you to create workbooks interactively.

Formula One is superb and its main competition is from Excel itself. Excel is a strong development tool and its worksheets can be embedded in other applications and controlled programmatically. Excel is the more powerful, but in comparison Formula One is small, nimble and royalty-free.

First Impression, the Suite's charting component, is updated to version 2.1. Not much has changed, mainly the support for double-byte character sets and Uniform Data Transfer.

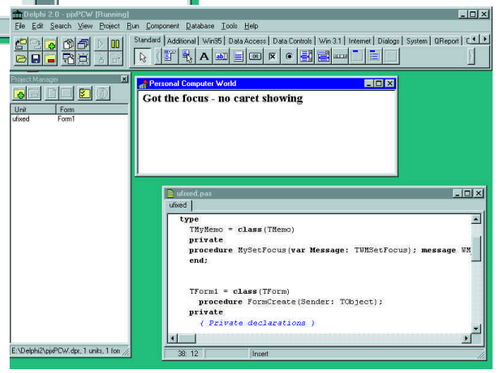
Visual Writer is a word processor control now at version 3.1. It's good, but not of the quality of Formula One. Under Windows 95 or NT, Visual Writer has to compete with the built-in Rich Text Control which does the same job of displaying formatted text with embedded bitmaps. Visual Writer does have some extra functions like fields, zooming, spell checking via the supplied Visual Speller control, print preview, and a ready-made button bar, status bar and ruler. It also has some quirks. Rich Text Format (RTF) is supported, but it prefers its own proprietary format. This is a disadvantage, especially since it will not accept .RTF as a valid format when bound to a document database. Also lacking is any kind of HTML support. For Windows 3.1 developers, though, Visual Writer or something like it is all but essential if you need to display formatted text. It's a shame the supplied 16-bit version is an

p316 ➢



Left Visual Writer is invaluable for 16-bit Windows but less useful in Windows 95

Below A solution to the flashing cursor problem. The TMemo object is subclassed, creating a component which is sufficiently general to be used many times over



OCX, as the VBX control type is more widely supported in Windows 3.1.

Carets and messages

The following question is asked by reader, Deborah Pate: *"How can I stop the cursor flashing in a TMemo component on a form with no other control that can accept the focus? Setting it to read-only does not help."*

This is a fair question, although I am not sure why you would *not* want the cursor flashing in a memo control that has the focus. Anyway, this is the kind of thing that should send you scurrying to the Windows API. One thing you must realise is that what most people call the cursor, Windows calls the caret. There are eight functions specifically concerned with this little flashing creature. For example, you can control the blink rate with SetCaretBlinkTime. Hiding the caret is just a matter of calling the right function. That is:

```
HideCaret(Memo1.handle);
```

The remaining problem is where to call the function. The obvious place is in the OnShow event method for the form but it doesn't work. The memo component receives a SetFocus message after the form shows and that helpfully reinstates the caret.

The OnPaint event does the trick but this is not the best solution. In certain circumstances the memo control can receive a SetFocus message without the form's OnPaint event firing, and back comes the caret. If you call HideCaret in enough places you can probably make it watertight, but it's not elegant programming.

The best answer is to trap the SetFocus message itself. To do this you need to subclass the TMemo object, giving it a custom message handler. You can either do this entirely within your application, or create a new custom memo component and install it on the component palette. For a one-off, the first approach is fine, the only snag being that because you create the control at runtime, you cannot place it visually or use the Object Inspector. How it works is shown in Fig 1.

Of course, you will also want to set other properties and perhaps write event code for the memo control, all of which you can do in code. Sledgehammer to crack a nut? Maybe. But once you have learnt how to subclass Delphi controls, many other problems can be easily solved.

You can also build up a library of customised components which can be used many times over. For example, you could create a memo with a Boolean ShowCaret property that turns caret display on or off. In the long term, the productivity gains are enormous.

Input Pro

Once upon a time, it was Aware/VBX. FarPoint has renamed this set of data-entry controls to the more natural Input Pro. It is an unglamorous collection, but is also one of the most useful for anyone doing data entry forms in Visual Basic or other ActiveX clients. A VBX version is also supplied.

There is not much extra functionality in Input Pro, as opposed to Aware/VBX. The main difference is the move to ActiveX. There are eight controls including currency, date and time, masked edit, and a memo control which overcomes the normal 64Kb limit. All are data-aware. The main purpose of InputPro is for validating data entry (never an easy task): its controls greatly simplify matters. For example, the DateTime control rejects invalid dates and times, can limit their range, and can auto-complete partial entries.

Fig 1 Trap that SetFocus

1. In the type section of the form unit, declare the following object:

```
TMyMemo = class(TMemo)
  private
  procedure MySetFocus(var Message: TWMSetFocus); message WM_SETFOCUS;
  end;
```

2. In the public declarations for TForm1, include:

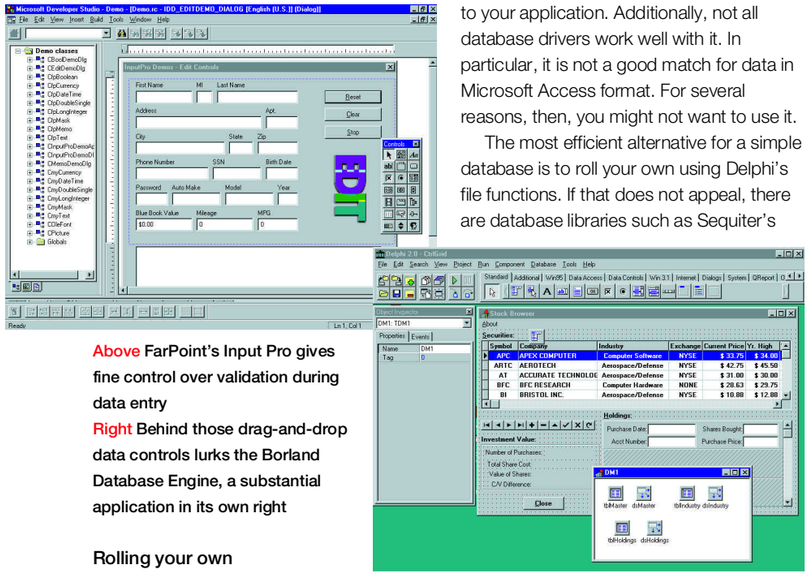
```
Memo1: TMyMemo;
```

3. In the implementation section, include:

```
Procedure TMyMemo.MySetFocus(var Message: TWMSetFocus);
begin
inherited; {call the default handler for this message}
hidecaret(self.handle); {hide the caret}
end;
```

4. In the FormCreate method, include:

```
Memo1 := TMyMemo.create(self);
Memo1.Parent := self;
HideCaret(Memo1.handle);
```

Above FarPoint's Input Pro gives fine control over validation during data entry

Right Behind those drag-and-drop data controls lurks the Borland Database Engine, a substantial application in its own right

Rolling your own

Reader Richard Hustwayte writes: *"My project will require some databases to be made — nothing complex like client/server but simple, flat-file databases. I have looked at two versions of Delphi: the standard version (about £70) and the desktop version (about £230). The latter version is advertised as having the Borland Database Engine. What is this? And if I don't have it, am I unable to create database applications?"*

All versions of Delphi come with the Borland Database Engine. This is a library of functions designed to simplify database work by acting as an intermediary between your application and the driver software which actually accesses the data. One benefit is that you can use data-aware components, so you can create simple database applications without writing any code. A number of database drivers are available for the BDE and it can also use ODBC drivers, the Windows standard for database access.

The BDE is good, but there is a cost involved. The BDE is a substantial piece of software and adds a considerable overhead to your application. Additionally, not all database drivers work well with it. In particular, it is not a good match for data in Microsoft Access format. For several reasons, then, you might not want to use it.

The most efficient alternative for a simple database is to roll your own using Delphi's file functions. If that does not appeal, there are database libraries such as Sequiter's CodePascal which provide a lightweight alternative. Then again, the BDE comes in the box and is fairly easy to use, so most Delphi developers do not look elsewhere.

Which Delphi book?

Darren Davies writes: *"I'm just about to buy Delphi Developer 2.0. I was wondering if you could recommend a book to learn how to program in this language? I've had quite a bit of experience with Pascal for DOS and object-oriented Pascal for Windows, but not much with visual programming."*

For someone with programming experience, a good choice is *Delphi 2.0 Unleashed* by Charles Calvert (SAMS). At 1,400 pages, it goes some way to compensating for Delphi's poor documentation.

A terminal problem

"I'm trying to get my new Terminal Program to automatically log in to a BBS. How do I get MSCOMM to wait for a login prompt before it enters the user details like Login name and Password?" asks Aaron Hodgson.

The MSCOMM custom control, which is similar in Visual Basic 3.0 and 4.0, offers two ways of intercepting data. The first technique involves a program loop which continually checks the receive buffer, a technique called "polling". Fig 2 shows what it looks like in pseudo-code. While it is a useful technique, it is difficult to write a well-structured application if it spends much time sitting in a loop like this. Another method is preferable, which is to use the OnComm event to respond to incoming data. This event fires whenever a communication error or event occurs. You can respond with a select case statement, like that shown in Fig 3 (page 319).

Your code should respond to all the

Fig 2 Pseudo-code for "polling"

```
Begin do loop
DoEvents or Sleep to allow windows to run other processes
Check InBufferCount property
If there is data, read input property and add to string buffer
Check buffer is not too full and correct if necessary
Check for time out, data complete, broken connection or other errors
End do loop
```

Java books

■ *Professional Java Fundamentals*
by Sly Cohen, Tom Mitchell and others

Most Java programmers are already skilled in another language: often C++. This book is aimed at that readership, providing a concise introduction to Java and focusing on its distinctive features. Beginning with a description of the Java language and object-oriented programming, it goes on to explain packages, threads and streams. Five chapters are devoted to the Abstract Window Toolkit, including a detailed explanation of various layout managers. The most advanced chapters cover networking, building libraries, implementing an application framework, and interfacing with C++.

There seem to be lots of poor Java titles around, and in contrast here is a knowledgeable and well-judged guide which complements rather than repeats what is easily obtainable online. Recommended.

■ *Using Java (Second Edition)*
by Joseph Weber and others

The flash on the cover states: "Covers new JDK 1.1 features" which is a bold claim since, at the time of writing, the JDK 1.1 was still in beta. You will find some useful material on JDBC database classes and a little on remote method invocation but, of course, much of JDK 1.1 is not actually included. What you get is over 1,000 pages which take you step-by-step through Java's tools, language, classes, applets and applications, graphics and layout, security and more. There is an emphasis on Sun's tools rather than third-party contributions, although the online version includes a chapter on different development environments.

Overall, *Using Java* is a thorough guide, although at times rather ponderous and unexciting. On the CD, you get online versions of four other titles covering JavaScript, Visual J++, CGI scripting and HTML, along with additional chapters and example Java applets. As a one-stop Java reference library, this book is hard to beat.

p319 ➢

Fig 4 WaitFor function

```
Function WaitFor(sWaitString As String, lTimeout As Long) As Integer

Dim lStartTime As Long
Dim sBuffer As String
Dim iOldThreshold as integer

lStartTime = Timer
iOldThreshold = Comm1.RThreshold
Comm1.RThreshold = 0
‘ prevents comEvReceive firing

Do
DoEvents ‘ or call Sleep API function
 If Comm1.InBufferCount > 0 Then
 sBuffer = sBuffer & Comm1.Input
 ‘ should check for buffer too large
 End If

 If InStr(sBuffer, sWaitString)  0 Then
 WaitFor = 0
 Exit Do
 End If

 If Timer >= (lStartTime + lTimeout) Then
 WaitFor = 1
 ‘ you can define constants and report errors
 ‘ using the return value
 Exit Do
 End If

Loop

Comm1.Rthreshold = iOldThreshold

End Function
```

Now you can write code like this:

```
If WaitFor(“login: “, 60) = 0 Then
‘ waits for up to 60 seconds
Comm1.Output = “qix” & Chr(13)
MsgBox “Successfully posted response”
Comm1.Rthreshhold = 1
‘ Enables comEvReceive event
Else
MsgBox “Login error”
Comm1.PortOpen = False ‘ Closes port
End If
```

possible events in order to trap communication errors. You also need to check that the string buffer is kept to a reasonable size. Using the CommEvent it is possible to write a reasonable communications program in Visual Basic, and there is an example called VbTerm that comes with Visual Basic.

Although the event-driven approach is better for most purposes, Aaron’s particular problem can easily be solved by polling. You can write a WaitFor function that doesn’t return until a particular piece of data has been sent, or until an error has occurred. An example of this is shown in Fig 4.

Note that if you have also written code to respond to the OnComm event, you need to ensure that events of the type comEvReceive do not fire when the WaitFor function is running. You can do this by setting the Rthreshold property to zero.

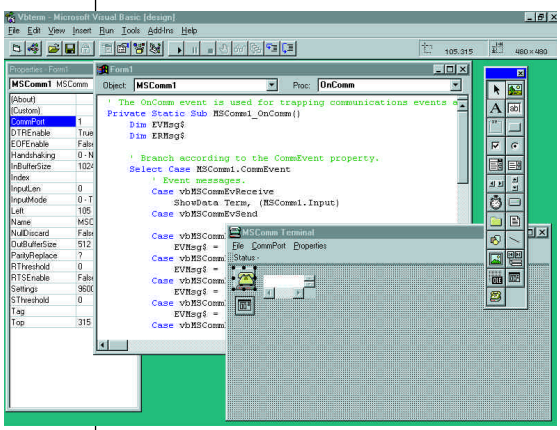
Finally, communications code is tricky, mainly because so many things can go wrong. At one extreme, poor lines and dropped connections cause difficulties, while the opposite problem is data coming in so fast that some part of your software cannot keep up. If this last problem occurs, Microsoft makes two recommendations. One is to extract data immediately the OnComm event fires, without bothering to check the type (see Fig 5). Also, the MSCOMM control may not always be satisfactory and as a last resort you can call the Windows API directly. This is well covered in the first edition of Daniel Appleman’s *Visual Basic Programmer’s Guide to the Windows API* but not in the second, 32-bit edition, although there is some material on the CD which accompanies the book.

Fig 3 A select case statement

```
Select Case Comm1.CommEvent
Case comEvReceive
  sBuffer = sBuffer & Comm1.Input
  ‘ or send to data processing function
Case comRxOver
  MsgBox “Error: receive buffer
 overflow”
Case comTxFull
  MsgBox “Error: transmit buffer full”
End Select
```

Fig 5 Extract data

```
Sub Comm1_OnComm ()
Static ReceiveBuffer As String
ReceiveBuffer = ReceiveBuffer &
Comm1.Input
Etc...
```



Above The VBTerm sample comes with Visual Basic and demonstrates the use of the MSCOMM control

PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address or at visual@pcw.vnu.co.uk

Visual Developer’s Suite Deal is £235 (plus VAT) from **Visual Components 01892 834343**
Input Pro (FarPoint) is £105 (plus VAT) from **Contemporary Software 01344 873434**

Professional Java Fundamentals, by Sly Cohen, Tom Mitchell and others is £32.49; ISBN 1-861000-38-3, published by Wrox Press.
Using Java (Second Edition) by Joseph Weber and others costs £56.49 (incl. VAT); ISBN 0-7897-0936-8, published by Que.
Delphi 2 Unleashed by Charles Calvert costs £54.95 (incl. VAT); ISBN 0-672-30858-4.
These books are available from **Computer Manuals 0121 706 6000**

Active service

Tim Anderson investigates the Active Platform – is it really new? Plus how to use resources in Delphi, new books reviewed, and a preview of the Visual Basic control creation edition.

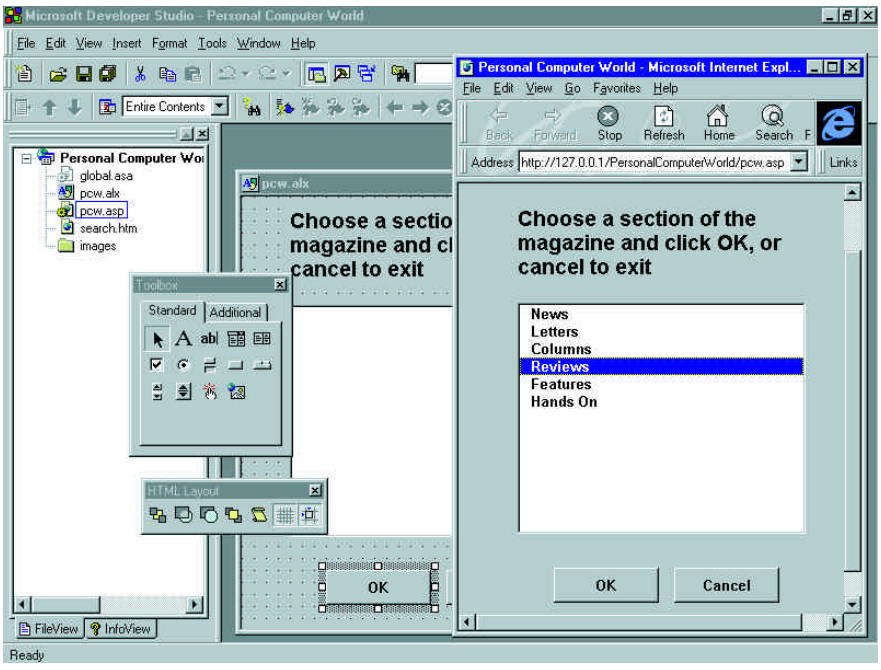
I'm sitting here looking at a sheaf of press releases and a stack of CDs which comprise the Microsoft Active Platform in its current, beta guise. The papers are an intricate display of verbal gymnastics: there are generous sprinklings of key buzzwords like open, standards-based, scaleable, multiple operating systems, and so on. The name Active Platform itself is a political statement. Sun calls Java a platform; Netscape Communications calls its browser a platform; others see the Network Computer as a platform. At stake is the question of who will be at the centre, and who will be satellites. Like all the best prima donnas, none of the main industry players wants to be anywhere less than centre stage.

Where there's an O there's an A

You might be forgiven for wondering where this new Active Platform comes from. Microsoft's publicity implies that a range of new technologies, such as the Active Desktop, the Active Server and Dynamic HTML, have emerged brand new and sparkling from a magician's black hat somewhere in Redmond.

The truth is more prosaic. For years Microsoft has been promising to rebuild Windows on an OLE foundation, and that strategy has not changed. In many cases Microsoft has simply replaced the word OLE with Active. So, ActiveX controls are OLE controls, OLE automation servers are now Active Servers, and similarly the OLE object model once known as Data Access Objects has become the ActiveX Data Object or ADO. With that in mind, here's a plain English summary of what is in the Active Platform.

1. Active Desktop This is essentially a web browser with support for HTML, VB Script, Java applets and ActiveX controls. In other words, it is Internet Explorer. Full implementation is in the forthcoming version 4.0, which is fully-integrated into the Windows shell.
2. Active Server This means that Internet Information Server can be controlled through what used to be called OLE automation.
3. Active Server Pages Here, Microsoft is referring to the ability to embed scripts, typically written in Visual Basic, into HTML web pages. Previously such scripts could only be executed by Internet Explorer on the client's PC. Now, a new tag lets you run the script on the server. Web sites have been doing this for years using CGI scripts, but this new approach is easier and removes the need to compile the script into a binary executable.
4. Dynamic HTML Code-named Trident, this is a set of extensions to HTML which implement much-needed features like layering and exact positioning. It provides an enhanced object model with more control over frames, tables and scripts.
5. Active Data Object Like Data Access Objects, this is a COM object model for database access. It hooks into ODBC for connectivity to a broad range of database servers.
6. Design-time ActiveX These are add-ins for Internet Studio which typically generate HTML and VB Script in response to user input while authoring a web page. You can



The Internet Studio project browser in file view (left), an ActiveX layout being designed (centre), and the resultant form at runtime in Internet Explorer

p296 ➢

Unmistakably a platform; but with the ActiveX Platform, things are less clear cut

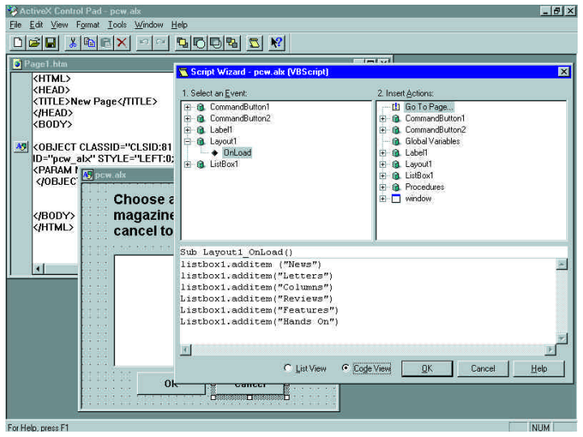
think of them as web page wizards. They are ActiveX controls but are not used at runtime and do not need to be downloaded to the client's computer.

7. HTML layout One of the most useful ActiveX controls is the HTML layout component, simply a container for other controls. Using the Active Control Pad or Internet Studio's layout editor, you can build forms which include scripts, much as you would with standalone Visual Basic.

The ActiveX Control Pad

Released without fanfare onto Microsoft's web site, the ActiveX Control Pad is an essential tool for authoring ActiveX applications. It combines a simple text editor with a VB-like form designer. The idea is that you open an HTML document in the editor, design a form known as an HTML Layout, and then insert it into the document. The HTML layout is itself an ActiveX control, but exists only as a container for other components. You can also insert ActiveX controls directly, without using a layout. The control pad generates a bit of HTML code using the OBJECT tag, including the long alphanumeric CLASSID which uniquely identifies each ActiveX component.

The control pad does not only handle the placement of controls. Using the script wizard, you can also write code to bring the form to life. The scripting language can be either VB Script or JavaScript, although the two cannot be mixed on one page. In list view, the script wizard will write code for you based on your response to dialogs, or you can choose code view and bang out code in the old-fashioned way.



The ActiveX control pad, showing an HTML document, a layout, and the script wizard

There are many advantages to using the control pad. One is that you can position controls precisely within the layout, something which you cannot do with pure HTML. Controls have a z-order too, so you can position one in front of another. The other plus is that a control's properties and methods are listed in the property editor and script wizard, so you do not need to look them up. Visual Basic programmers will soon feel at home. A similar tool is in Internet Studio, where it is called the HTML layout editor.

The biggest problem is that the control pad does no syntax checking and has no debugger — a sure sign of immaturity. Internet Explorer will report errors in your code, but otherwise you are reduced to tricks like throwing up message boxes to check the status of variables. The other problem is that ActiveX layouts currently work only with Internet Explorer 3.0. No surprise there.

Will it work?

The usefulness or otherwise of Microsoft's new web initiative depends on which hat you wear. For general Windows developers, this is significant. Internet Studio, the tool that brings all these gizmos together, is a viable alternative to Visual Basic and Access. You can design forms, write VB code, and simply have your final application run within Internet Explorer rather than directly from the Windows desktop.

As Windows evolves, that last distinction will become increasingly blurred. The advantages are that your application can be Intranet-ready and database independent. On an Intranet, you have full control over whether code is executed on the client or on the server. Assuming Windows remains popular, I see this kind of approach as gradually replacing existing development techniques.

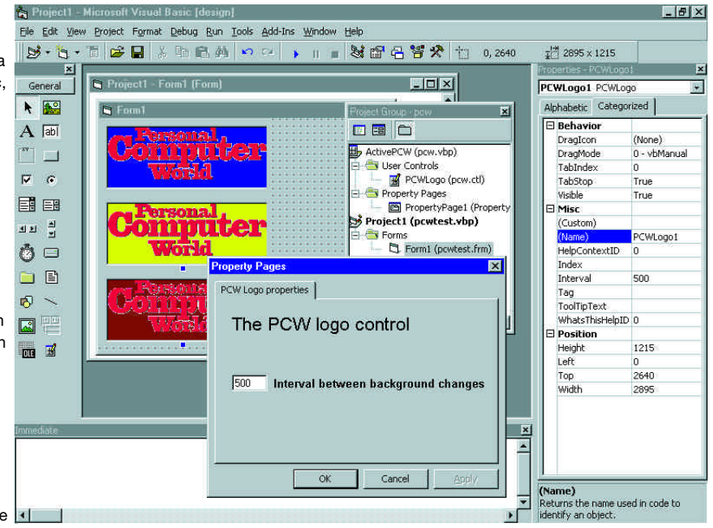
What about Web developers? It is these that Microsoft is courting most visibly, with its "site builder" initiative. But success is far from assured. Microsoft can do what it likes with Windows, but does not own the web. The grave weakness of its Active Platform is that, despite noises to the contrary, it is not a cross-platform initiative. There is no problem with the server-side aspect, since the server can do what it likes as long as it delivers HTML that the browser can understand. The problem is with the Active Desktop. ActiveX controls, remember, are binary executables which run natively on the client's computer. If you want to create an ActiveX control which runs on, say, Windows, the Macintosh and Unix, then you must create three separate executable files. Even if Microsoft delivers what it promises, versions of Internet Explorer for these platforms, it is hard to see this strategy working.

By contrast, a Java applet runs on any platform for which a Java Virtual Machine exists. That means Sun's Java Beans model holds all the cross-platform aces. Java applets can accomplish many of the same tasks as ActiveX components. Performance can be poor, but just-in-time compilers and eventually Java-based operating systems will crack that problem. Microsoft is making it enticingly easy to create web sites built with ActiveX controls, but such sites will to some extent shut out non-Windows browsers. If that drives more people to use Windows, Microsoft wins. But if these factors lead to Java rather than ActiveX dominating the Internet, the popularity of Windows itself will inevitably decline. The stakes are high.

Visual Basic Control Creation Edition

Unlike Internet Studio or the ActiveX control pad, the VB Control Creation Edition is not just for web development. As its name implies, it is a tool for creating ActiveX controls in Visual Basic, and these controls can then be used in any Windows development tool or document capable of hosting ActiveX, formerly known as OCX controls. In its determination to reinforce the ActiveX standard, Microsoft is making the control edition a free download, both the beta and final versions. Incidentally, it also offers a preview of what the VB 5.0 interface may look like when it emerges.

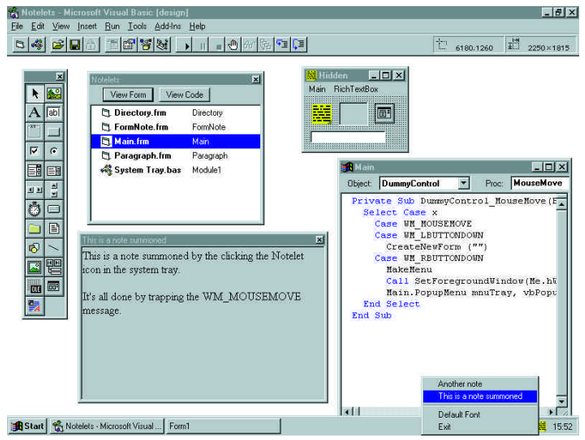
Since version 4.0 Visual Basic has been able to create OLE automation servers. You can declare an object class in a VB project, and then have other applications create objects of that class. Borland's Delphi 2.0 is similarly capable. The one piece missing in both products is the ability to create OLE objects that have a visual interface, or in other words, ActiveX controls. That gap has now been plugged. With the control creation edition, you can develop ActiveX components that can be installed on the component palette in products like Visual Basic, Access and Delphi. It is a great step forward, the main snag being that in this version, compilation to native code is not possible, so performance will not match ActiveX controls written in C++. VB controls can be very small, but require a substantial runtime library which makes distribution awkward. Microsoft now calls this the VB Virtual Machine. The implication is that a VM for Visual Basic may be implemented on more than one platform, although Microsoft has not stated this explicitly. Such a move would make ActiveX a more plausible cross-platform contender.



A preview of Visual Basic 5.0. The Control Creation Edition at last makes it easy to write ActiveX components

To test the control creation edition, I built a simple control. Using an image control and a timer, I displayed the *PCW* logo on a form. With one line of code I made the logo's background colour change whenever the timer event fired. Next, I used the Interface wizard to choose which properties and methods to expose, including a custom property to set the timer interval. The property page wizard created a standard property page, and finally Make OCX built the control.

Nobody can now say that creating an ActiveX control is difficult. The main flaw in the VB control creation edition is not technology, but human fallibility. Creating a control is easy; but creating a good control still requires skill. The documentation observes how important it is to maintain a consistent interface when controls are developed, and warns that a poorly-implemented control can be a security risk even without malicious intent on the part of its developer. For example, if a method is exposed that enables a named file to be created on the user's hard disk, the control is not safe for scripting. Considering the number of VB developers, both professional and hobbyist, mistakes are inevitable.



Creating full system tray apps with native Visual Basic 4. See the full code on the cover CD

Delphi

Delphi and Windows resources

Delphi can use standard Windows resource files (with a .RES extension) and indeed there are occasions when this might be essential: creating a screensaver for Windows NT, for example. It is yet another of those areas which Borland has scarcely bothered to document. Bizarrely, there is documentation in WINAPI.HLP, supplied with Delphi, that covers the Microsoft command-line resource compiler, but not for the Borland resource compiler actually supplied. You didn't know Delphi comes with a resource compiler? It does, and it is the executable called BRCC.EXE or BRCC32.EXE, the 16- and 32-bit versions respectively. The versions called BRC.EXE and BRC32.EXE are shells which are able to call both the resource compiler and the resource linker, RLINK, to bind a resource to an executable — but you do not need to know this since Delphi will do it for you.

To find out how these programs work, run them from a command line without parameters and the options are displayed.

What Delphi does not have is a resource editor. Simple resource scripts can be created by hand, otherwise you will want to use an editor such as the one distributed

p299

with Borland's C++ products. Using resources in Delphi takes several steps:
1. Create a resource script and compile it to a .RES file.
2. In your Delphi application, include the compiler directive:

```
{$R MYRES.RES}
```

where MYRES.RES is the name of your resource file. A good place to put it is in the project source below the similar directive {$R *.RES} which Delphi includes by default in all projects. The reason is that the application icon is stored in a .RES file of the same name as the project. It is best not to edit this generated resource file, since Delphi may overwrite your work.
3. Your Delphi code can now load these resources using API functions. Here is a simple example. The following resource script contains a string table with one string:

```
STRINGTABLEBEGIN1, "I wandered
lonely as a cloud"END
```

Save this as TEST.RC, compile it using BRCC to TEST.RES, and then include it in a Delphi project. Now you can retrieve the string in your Delphi application as follows:

```
lpzTest := stralloc(26);
LoadString(hInstance,1,lpzTest,25);
```

where lpzTest is declared as a pChar. LoadString's second parameter is the ID of the string to load, often replaced with a constant for clarity, and the last parameter is the maximum length of the string to retrieve.

Visual Basic

More about the System Tray

James Talbut writes:

"You mention the usage of the Shell_NotifyIcon function and state that it is not possible to use the messages without additional software. But you can. Essentially you create a hidden control on your form and use an unrequired message for controlling it."

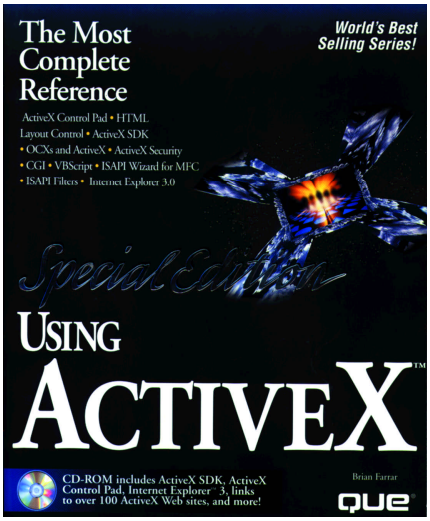
The system tray is controlled by an API call Shell_NotifyIcon, which takes a pointer to a NOTIFYICONDATA record as one of its parameters. This record includes fields for a window handle and a message identifier, the idea being that Windows sends that message to the specified window when
the user clicks on an icon in the tray.

In C++ or Delphi you would use a custom message handler, but VB does not offer that facility. The workaround is to hijack an existing message handler, and James suggests using a hidden picture box and the WM_MOUSEMOVE message. Then, you can write code in the MouseMove event that will respond to system tray events.

It works, and James has written an example notelet application, which is on the cover CD. It lets you store notes which pop up when you right-click an icon in the tray. Thanks, James – you have won a book token for your efforts.

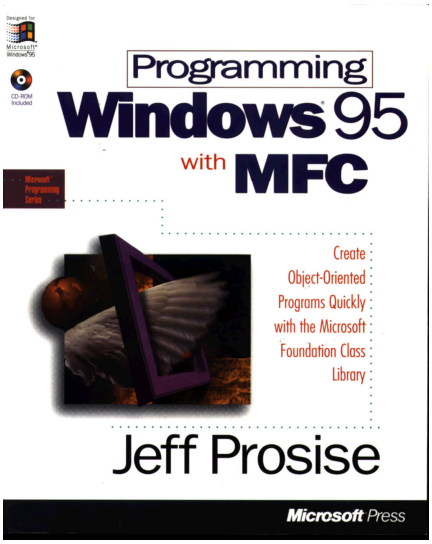
Books for Visual Programming

Using ActiveX by Brian Farrar
This is nearly very good. Aimed at those considering doing the web the Microsoft way, it presents all the main elements clearly and concisely, with examples. The book covers VB Script, ActiveX technology, the Control Pad, the Internet Control Pack, Internet Information Server and its ISAPI interface, and CGI scripting. It is fine as an introduction and overview, but does not go into enough depth to merit the "using" part of its title. For example, ActiveX security issues are skated over in a couple of pages. To be fair, Microsoft's ActiveX SDK, included on the supplied CD, does give the required detail; but most readers will have it already from another source. Buy this for an excellent overview, but expect to need further help very soon afterwards.



Using VBScript by Ron Schwarz and Ibrahim Malluf
This title is both longer and more tightly focused than its companion, Using ActiveX, in the same series. Without assuming much prior knowledge, the authors show how to program web pages using VB Script, touching on related areas like ActiveX and SQL Server web extensions. Considerable space is given to HTML itself, including an appendix documenting all HTML tags supported by Internet Explorer 3.0. There is a CD with all the examples from the book, and as a bonus, the full text of another Que title, Using Visual Basic 4.0. It is a nice extra, but ironically none of the web pages on the CD are well designed. Overall, it is a good introduction to VB Script, but do not expect it to answer all your Web queries.

Programming Windows 95 with MFC by Jeff Prossie



You have to respect someone who knows his limitations. Jeff Prossie is not a database programmer, nor is he an OLE enthusiast. "Certain parts of OLE are promising," he says, "but the OLE documents protocol is overly difficult to implement and of limited value in the real world." That explains why his book on Microsoft's Foundation Classes, the leading C++ Windows class library, covers neither MFC's database classes, nor OLE in any form. Instead, he gives a nuts-and-bolts description of how to program with MFC, starting with "Hello World" and progressing to documents, views, common controls and multi-threaded development. It is a valuable book, since most other tutorials focus more on using Visual C++ and its wizards, than on MFC itself. Look elsewhere for ActiveX, web development or database work, but buy this book to learn the fundamentals of Windows development using MFC.

PCW Details

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address or at **visual@pcw.vnu.co.uk**

Programming Windows 95 with MFC by Jeff Prossie (Microsoft Press), book and CD, £49.95 inc VAT.
Using ActiveX by Brain Farrar (Que), book and CD, £37.99 inc VAT.
Using VB Script by Ron Schwarz and Ibrahim Malluf (Que), book and CD, £46.99 inc VAT.

Spare **parts**

Tim Anderson compares components for Basic and Delphi. Plus, for those who can't see the wood for the trees there's a guide to choosing a visual programming package .

Visual programming means dropping components into your application and making them go by setting properties and calling methods. That, at least, is the plan and here is an evaluation of some recent components. Your views matter most though, so please let me know which components work well or badly for you.

Crystal Reports 5.0

Crystal Reports is hard to avoid, being widely bundled with products like Visual Basic and Visual dBase. Seagate naturally hopes that users of these bundled versions will want to upgrade. Version 5.0 is the latest release. The Standard version supports most desktop database formats like dBase, Access and Paradox, while the Professional version adds full ODBC and various native SQL formats.

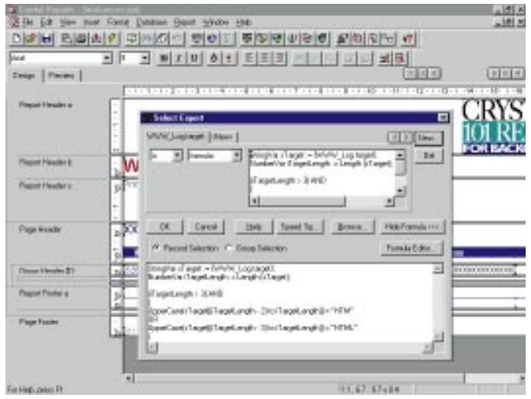
There is a new interface for designing reports, with better drawing features and in-place OLE editing. Of most interest to developers is the new sub-report feature, which enables you to insert a report within a report. Normally, this would be linked to the main report for displaying child records, but it can also display unrelated data. Another new feature enables you to export HTML for adding to web sites.

Crystal is a powerful tool and has components for most development languages including 16- and 32-bit Visual Basic, Delphi and C++. But I do have reservations. One is the sheer size of the product: the main print engine is now over 3Mb; another is that Crystal has its own formula language and although reasonably capable, it is ugly and old-fashioned. Nevertheless, version 5.0 is a significant upgrade and developers with better things to do than write their own report engine will find it invaluable.

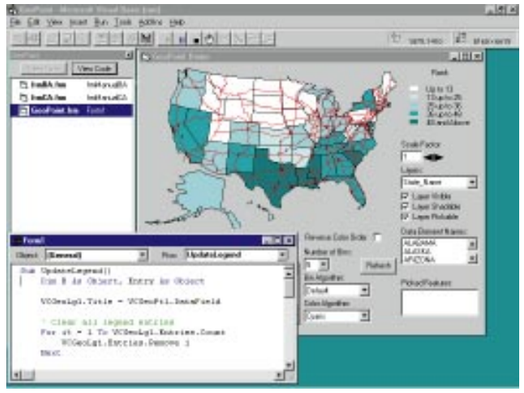
GeoPoint 1.0

Visual Components is responsible for some of the best ActiveX components around, including Formula 1 and Visual Writer. GeoPoint is a more specialist control. It displays maps in MapInfo or Autocad format. By using it alongside the separate Legend control, you can programmatically control the text and shading of each area of the map. A technique called "binning" lets you categorise data into ranges, and then shade the map accordingly: a typical example would be a display of sales performance by region. The user can also select an area of the map by clicking, so the application can show related data. GeoPoint can be bound to a data control to display your data.

This useful component is spoilt by its presentation. There is no printed manual and the online documentation is poor. The other snag is that the few supplied maps, heavily biased to the USA, are not likely to be what you want. That means purchasing add-on maps, or buying MapInfo or Autocad to create your own. Making full use of GeoPoint will be expensive.



Above **The Crystal Select Expert lets you create custom fields using the formula editor**

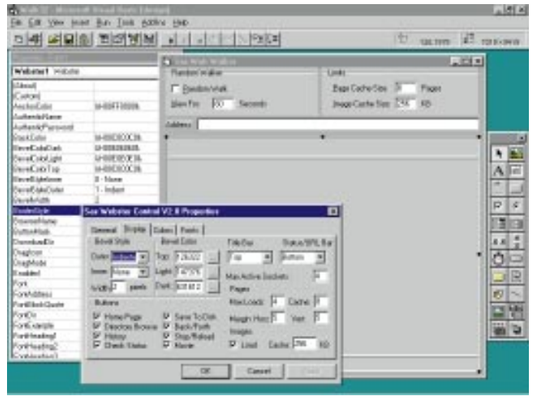


Right **GeoPoint is a neat tool with which to analyse geographic data, but its USA map will not be of much use in the UK**

Sax Webster or Internet Explorer?

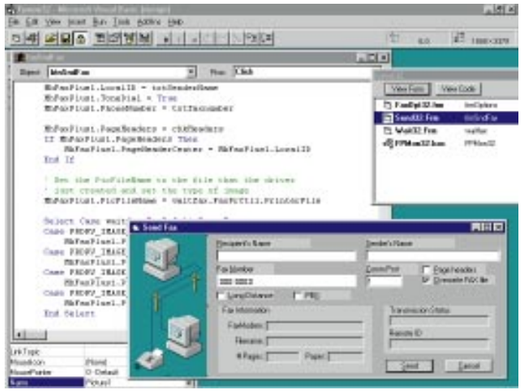
The Sax Webster control displays HTML documents. Now at version 2.0, it comes as 16- or 32-bit OCX controls that you can drop into your application.

It is easy to use. You can, for example, display a web page by setting the PageURL property. The main change from the first Webster control is the HTML version supported, now version 3.0 but without frames. It works well and may be useful on 16-bit systems or where a small memory footprint is required. Otherwise, on 32-bit Windows a better option is Microsoft's freely available Internet Explorer 3.0. In Visual Basic 4.0, open the Custom Control dialogue and check Microsoft Internet Controls. This installs the WebBrowser component which is the HTML display part of Internet Explorer. It is just as easy to use as Webster, and far richer in terms of HTML support. You will need to get hold of the Internet Explorer object model, which is part of the ActiveX SDK available on the Microsoft web site.



Left **Sax Webster: it works, but why not use Internet Explorer instead?**

Below **FaxPlus: does it have to be so complicated?**



Microhelp Fax Plus

Fax software may not be exciting but it is exceptionally useful, at least until the whole world gets webbed.

Fax Plus 2.0 is the new 32-bit version of Microhelp's Fax add-on. It is designed for Windows 3.1 and 95, but not NT. It consists of several controls, including a fax control that handles communications, and a FaxImage control for creating and modifying fax bitmaps. There is a printer driver and control which lets you send faxes by printing from any Windows application. The Fax Plus driver creates a fax image and then fires a StartDoc event in the printer control, so that your code can handle sending the fax.

Unfortunately though, using FaxPlus is not as easy as it should be. In part this is because of fickle telephone lines and diverse hardware that turn faxing into a trouble-prone business. Other problems are down to FaxPlus itself, which is awkward to code and not entirely bug-free. For instance, at the time of writing, the VBX version is unable to correctly convert text files to fax images. Still, it beats trying to write your own fax driver.

Microhelp VB Viewer 2.0

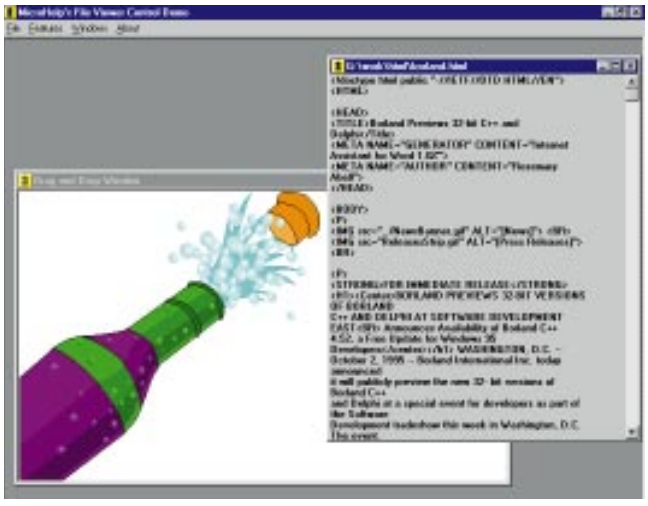
VB Viewer is a product of limited ambitions. Drop it onto a form and you can use it to display files of around 30 different types, including multimedia files. With text documents you can search for text and

p323 ➢

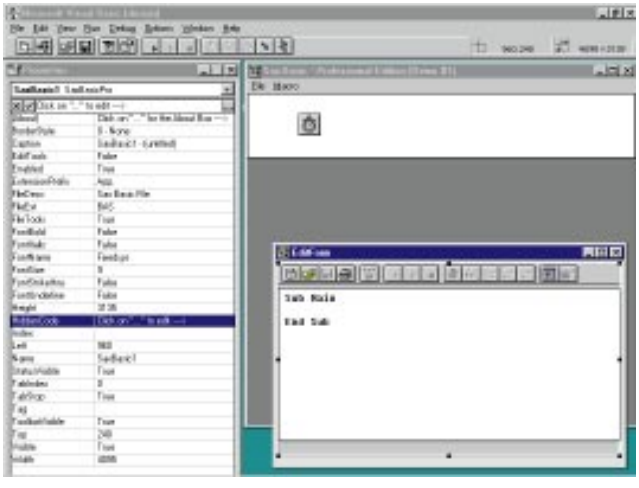
How to choose a visual language

Shaun Nicolson writes: "*I am considering buying a visual programming package but cannot decide which one. I am considering Microsoft's Visual Basic and Visual C++. The language I choose would have to produce network applications. What are the pro's and con's of these?*"

1. Ease of use. This is where Visual Basic scores highly, since you can have a simple utility up and running very fast. That does not mean VB will be the easiest for a large, complex application, since many other factors will then come into play. Delphi also scores well, while Visual C++ is hard to learn with limited visual tools.
2. Performance. This is where languages that compile to native executable code, like C++ or Delphi, generally win over interpreted languages like VB, FoxPro or Java. Database processing speed should be judged separately as all products use fully-compiled database engines. In some applications, performance is not an issue, or is determined more by factors like graphics or hard disk speed rather than the language used.
3. Power. The developer's nightmare is to spend months on a project, only to find that some intractable problem means that it cannot be completed with the current tool. Visual Basic is vulnerable since some features of Windows like callback functions or custom message handling are not available. There is usually a way around it by using a custom control or DLL but these must be written in another language. Version 5.0 should solve some of these problems. If you dread brick walls, C++ is the safest option, with Delphi a close second.
4. Database engine. Most languages have a native database engine, along with the ability to connect to other databases via SQL libraries or ODBC. If you know which database you will be working with, good connectivity is the first thing to check.
5. Availability of add-ons. This is where Visual Basic scores best. Most VBX and ActiveX controls are designed for VB, and may not work well elsewhere. There are also plenty of code libraries for C and C++, but native add-ons for other languages are more limited.
6. Reusable code. To protect your investment, you want to write code that will be reusable in future projects, perhaps even on other computer platforms. This is one of the benefits of object orientation, with Delphi scoring well, C++ fairly well, and Visual Basic less well. Best of all is Java, which forces you to write object-orientated code and runs on multiple platforms.



Left VB Viewer can manage a picture, but struggles to display HTML



Below A language within a language, Sax Basic lets you deliver programmable applications

copy to the clipboard. But overall, VB Viewer is a disappointment. One let-down is that formatted documents in word processor formats are displayed as plain text only, unlike the much better QuickView utility that comes free with Windows 95. Some basic formats like HTML and Rich Text Format are not supported at all. You can set VB Viewer to use QuickView viewers, but files then appear in the QuickView window rather than embedded in your form.

Sax Basic Engine

One way to impress users is to supply an application with its own macro language, like Excel or Word. The Sax Basic Engine lets you do just that. The control has its own IDE, so getting started takes little more than placing it onto a form. The language itself is compatible with Visual Basic for Applications, with excellent support for OLE automation and class modules.

To make Sax Basic useful, you need to extend its language to communicate with your application. The way to do this differs, depending on whether you use the VBX or OCX version. With the VBX, you can add keywords that fire an event called AppExec. You can then write code using Select Case to interpret the command. The OCX version has a more elegant solution. You write extensions in a VB class module, and then add them to Sax Basic using the control's AddExtension method. If your application needs a macro language, Sax Basic is ideal and a lot cheaper than licensing the genuine VBA from Microsoft.

Visual Basic

MT Emms writes: "*Using Paint I've created four BMP files. Each is divided into four sectors, the other three being left transparent. I have written a program to merge these bitmaps into one but the last one dominates — in other words, the transparent sectors are not transparent. It was simple on the Mac and Archimedes; surely VB should be able to cope?*"

Visual Basic *can* cope, but it is not as simple as it might be. The secret in this case is to use the PaintPicture method, on either a form or a picture box. The syntax for PaintPicture is:

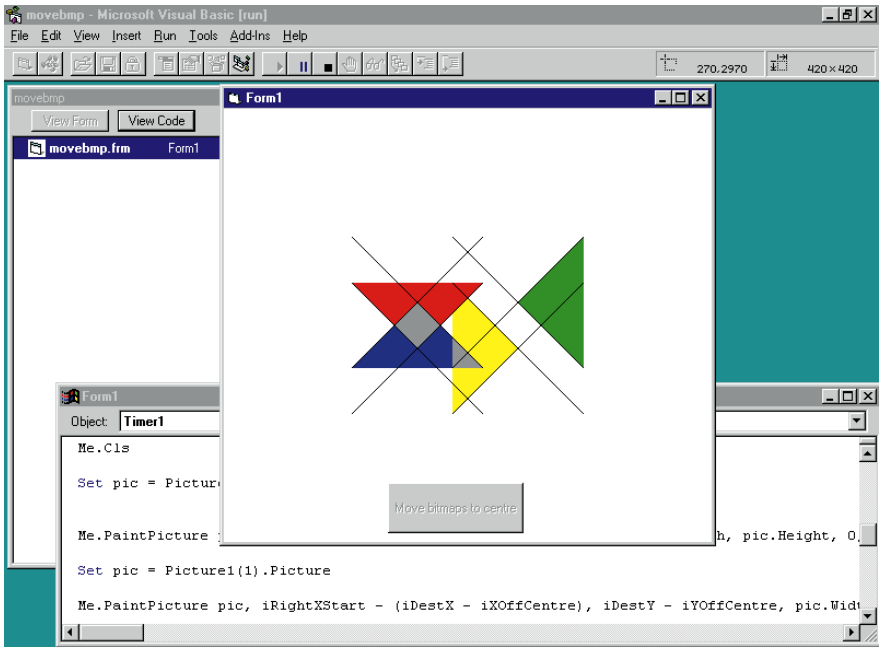
```
object.PaintPicture picture, x1,
 y1, width1, height1, x2, y2,
 width2, height2, opcode
```

The final parameter is a long value that defines a bit-wise operation which is performed on the picture as it is drawn. As the VB manual remarks, you can find a list of these operators in the BitBlt topic in the Windows SDK help.

The easy way to use them is to define them as constants in your VB application. For example:

```
Const SRCAND = &H8800C6  ' (DWORD)
 dest = source AND dest
Const SRCCOPY = &HCC0020 ' (DWORD)
 dest = source
```

If you then call PaintPicture with the SRCAND constant value, the bitmaps will merge in the way Mr Emms requires. Yes, it is more like programming in C than in Visual Basic, but at least it does the job. An example application is on our cover-



Merging bitmaps using VB and the SRCAND bitwise operator

p324 ➢

mounted CD, which also shows how to move the bitmaps across a form for an animated effect.

Delphi departure

In October, Anders Hejlsberg, the chief architect of Delphi, announced his departure from Borland for the safe pastures of Microsoft. Zack Urlocker, another key member of the Delphi team, has pointed out that "the architectural work that Anders covers is complete for Delphi 97. Anders' departure won't affect the shipping date or features going forward."

Even so, Anders is widely seen as the man without whom Delphi would never have happened, so his move is significant news. If anyone can knock VB into better shape as an OO language, he must be the man. Although Delphi is as good as ever, this weakens the case for migration from Microsoft tools. Personally, I hope that Borland can sustain Delphi's momentum, as it still delivers the best combination of rapid development, power and performance.

Dear Santa…

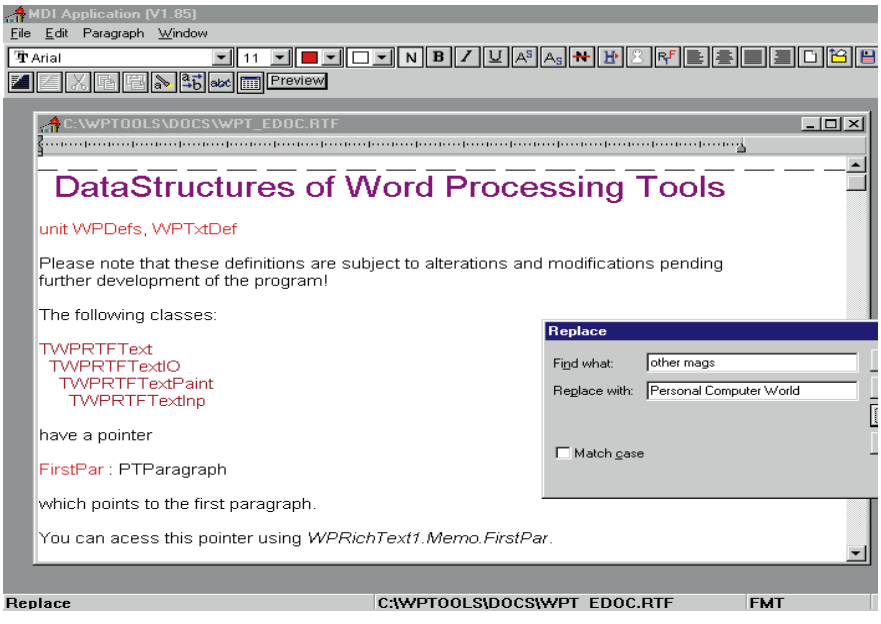
Sensible developers want an easy life. That means fast application building, reusable code, blinding performance and results that run everywhere. Well Santa, it seems you have a habit of giving with one hand and taking back with the other. Last year I asked for an end to the OS wars: a fanciful request, perhaps, but twelve months on and Java may provide an answer. Except that (dear Santa), we need easier, richer interface building, better performance, and decent support for platforms which Java finds difficult, like Windows 3.1 and the Macintosh. In the meantime, even developers who fix upon Windows have three versions with which to contend.

Forget the platform, then, let's look at the tools. First, there's Visual Basic, still the most popular all-round Windows language. Last year's wish-list included a compiler and better OO. The signs are that VB 5.0 delivers some of that, although it will never have the elegance of Delphi's component library. But Office 97 and VBA 5.0 are great news for developers of Office solutions. Thanks, Santa. And thanks for Optima ++, which is real visual development for C++ at last.

While I have your attention, there are a few things I'd like in my stocking for next year. Top of the list has to be a faster, better-organised internet. The web is irresistible for developers, both for technical support and as a platform in itself. But it has to get quicker and more reliable. *Please.*

Second, an un-present: Windows 3.1, please take it away. It's as bad as DOS, but worse, because people with working Windows 3.1 installations see no reason to change. I understand their point of view, but for developers this is a disaster. Develop two versions, with all the extra costs? Develop 16-bit only and waste all the advantages in 95 and NT? Develop 32-bit only and forget half the market? Hmmm, did I hear someone mention Java?

Third, I'd like better tools for troubleshooting OLE, ActiveX, COM, call it what you will. It's funny how quickly a Visual Basic 4.0 installation produces an "object server not correctly registered" message when you install custom controls. It means a registry problem and there's no easy way to fix it. This highlights a problem that will get worse if ActiveX continues to grow.



WP Tools is a fully-featured, shareware-rich text control for 16- or 32-bit Delphi

Word processing tools

Some months ago I mentioned a shareware product called WP Tools, a native Rich Text control for Delphi. On closer inspection, I am impressed. The feature list is good, with support for merge fields, graphics, tables and hyperlinks. The range of controls has a lightweight rich text label and a data-aware text box as well as the usual word processor, toolbar and status bar components. In tests, it has proved fairly reliable, though not entirely bug-free.

The advantage of WP Tools is that as a native VCL supplied with source, you can track down bugs and amend the code if you can work out what is going wrong. Another benefit is richer functionality. You can access the data structures for both text and formatting, giving a fine degree of control. The Finder class offers sophisticated search and replace, including formatting properties. You can print to a canvas control in order to implement page preview.

Performance is good, on a par with rivals like Visual Writer, AllText and HighEdit. If you are developing for 16-bit Windows, a custom component is all but essential for working with formatted text, while even in Windows 95 and NT it has advantages over the built-in rich text control.

There are problems. This is shareware, and the documentation is unclear. Advanced users need to be comfortable with such things as streaming and pointers, as WP Tools uses them extensively. To succeed with this product, you must be willing to pore over the source and not be put off by the odd mixture of English and German comments in the code. The extra effort and risk is rewarded by a product that works rather well.

•PCW Contacts

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

Components listed below are available from:
Contemporary Software 01344 873343; Grey Matter 01364 654100; and QBS 0181 956 8000.
Sax Webster £110 for the 16- or 32-bit version (£175 for both), plus VAT.
VB Viewer £110 (plus VAT)
FaxPlus £195 (plus VAT)
Seagate Crystal Reports 5.0 Standard £199, Professional £299 (both plus VAT)
Sax Basic 3.0 Pro £345 (plus VAT)

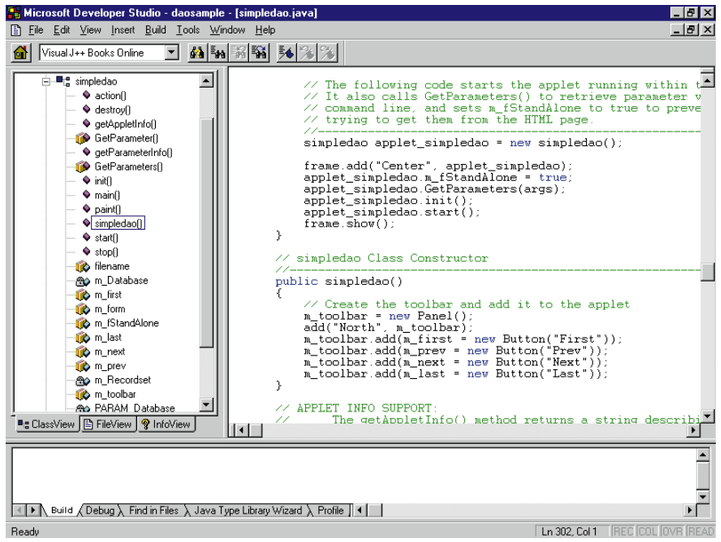
GeoPoint 1.0 costs £195 (plus VAT) from **Visual Components 01892 834343**
WP Tools is $119 to register, available via CompuServe, or from the web at **members.aol.com/JZIERSCH/wptools**

Future **threat**

What's behind Visual J++? Might it overtake C++ as a mainstream Windows application? Tim Anderson talks to Microsoft. Plus, screensavers in Delphi, and a free MSDN sample.

Microsoft's Visual Java tool is now out (see our review in "First Impressions"). But what is the company doing with a language that threatens Windows desktop dominance? Another consideration is that Sun's proposed Java Beans component model is at odds with Microsoft's ActiveX strategy.

I interviewed Microsoft's development manager, Greg DeMichillie, to find out more.



Visual J++ integrates seamlessly into Windows, but will other Java players accept Microsoft's ActiveX standards?

Visual J++

PCW: *I was disappointed by the lack of a visual environment for building an interface.*
DeMichillie: "I totally agree. Our long-term direction *is* towards graphical interface builders but the reason we haven't got this is because of work going on with class libraries. The question is whether the Abstract Window Toolkit is the long-term windowing model, or whether it will it be an alternative? I'll be back in less than a year talking about a new version of Visual J++."

DeMichillie: Aiming for large-scale development in Visual J++

PCW: *What are the problems with the AWT library?*
De Michillie: "The first question is whether AWT will continue to only do things that can be done on 19 Unix variations plus the Mac, plus Windows 3.1, plus NT, plus 95. Second, AWT was developed by many different people and that comes through in the APIs that are exposed. There are different designs and they don't mesh well. There are problems with layout, which is entirely code-based. That means when I lay out a form, the layout is stored in the code for the form's class. There's no separation of the code from the data. Maybe these points are addressable, but the larger architectural problems are more difficult.

"I think AWT was rushed out early. A lot of the fundamental Java technology was ready to go; the byte codes, the compiler and AWT got rushed. We support AWT because there are no viable alternatives."

PCW: *How would you envisage your class library developing? Would you implement Windows-specific features, or go for compatibility?*
DeMichillie: "My personal view is that the least common denominator solutions are not ultimately compelling. We want to expose all the richness and functionality of Windows but we want to do so in a way that enables us to port to other platforms.

"For example, take Direct3D and DirectX, our multimedia systems. Those take advantage of high-performance Windows graphics cards, but the API is generic enough to implement on other systems. Or database access — there's no reason database access APIs would only be on the Windows platform."

PCW: *Is Microsoft happy to see Sun controlling the language, or would you like to see an ANSI Java, or something like that?*
DeMichillie: "We don't necessarily need an ANSI committee, Sun has control over what is considered standard Java. But there are a number of vendors working on class libraries independent of Sun and over which Sun has no influence.

"I would expect Sun to be keenly involved in things like the byte code format, but I don't think class libraries are really one of those areas. In our relationship with Sun, we're competitive in many areas and I make no apology for that. But having said that, there are going to be huge areas of commonality. We don't want to see byte code format proliferation. I think the Java language will evolve."

PCW: *What about the Java Beans proposals? Do they fall into the area that is competitive?*
DeMichillie: "It's difficult to say, simply because Beans is just so vaguely defined at this point. We want to make sure that anything Beans does works well with COM.

"Our overriding goal is to make sure that Java developers have access to the thousands and thousands of COM components that already ship. ActiveX controls and OLE controls form the most successful component software market."

PCW: *Why have you hooked into Internet Explorer (IE) and not made your product browser-independent?*
DeMichillie: "We're not directly hooked into IE. We're hooked directly into MS's implementation of the Java Virtual Machine, which currently is hosted inside IE. But because the interpreter is itself an ActiveX control, that VM could be hosted inside any executable. The first reason is that the VM offers COM support and ActiveX support, and second, the VM supports a new set of debug interfaces. I would personally love Netscape to adopt the Microsoft VM so we could cut down on the number of VMs that are out there."

PCW: *At some future point, might Visual J++ be able to compete with Visual C++ in creating mainstream Windows applications?*
DeMichillie: "Sure. My ultimate goal is for large-scale development to be possible in J++. The growth in Java will come at the expense of C++. People have now dealt with C++ for a number of years and have seen some aspects that are more complicated than they might like. Java offers a simplification that is very appealing. As the tools evolve, Java will be able to do many of the things that now would need C++.

"It's important to distinguish between component builders and component users. The majority of ActiveX controls out there are written with C++. VB 5.0 will also create controls. It will be interesting to see where the component builders go. Component users gravitate towards VB and will eventually gravitate towards VJ++ as these very GUI-based, RAD-like tools appear."

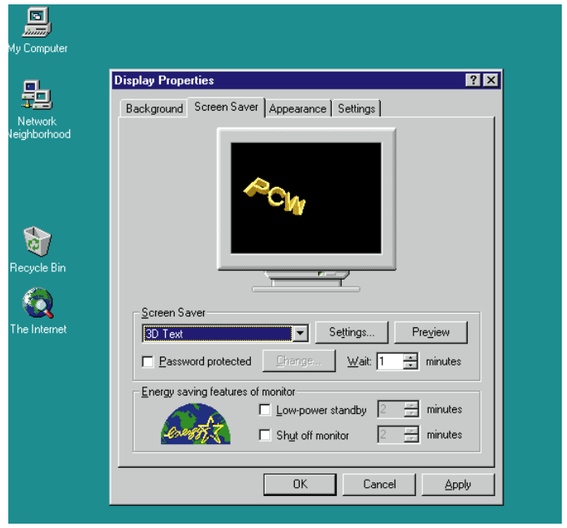
PCW: *If the lowest common denominator is not a long-term solution, does that mean cross-platform isn't either?*
DeMichillie: "No. There will be a core subset that's the same everywhere. But there will also be extra capabilities, even in Java, where one platform has an extra class and another doesn't. For example, there are a number of capability differences between Windows and the Macintosh. Do you really want to restrict the class library to only those that are common? Or do you want a class library that has the room to contain components that maybe work on three or four platforms, and other components that work on a different three or four platforms? Microsoft understands that the market is not just Windows, but includes Macintosh as well as Unix. But it does not follow that you are only going to do things that can be implemented on every platform."

Books for Visual Programming

Visual J++ by Charles Wood
The best and worst thing about this book is that it exists. It was on sale before Visual J++ was officially released in the UK and unfortunately it shows signs of haste. Nearly a third of the book is a list of Java methods — something you can get from online help. There is only brief coverage of ActiveX and COM and the author refers us to his forthcoming, more advanced book on the subject. Data Access Objects, one of the key selling points of Visual J++, are hardly mentioned. The rather important resource wizard (which lets you convert Windows menus and dialogues to Java) is skimmed over as being "beyond the scope of this book". What's left is an unexceptional general introduction to Java programming. It's not really the author's fault, since he was working with a beta product. But my message to you is, beware — the first books to come out for a new product are often not the best.

DELPHI

Anyone who writes a screensaver must have time on their hands. Screen burn is not common now, and in any case, perfectly functional savers are supplied with Windows. But they are fun! At least,



The Windows 95 screensaver control panel looks slick, but increases the work for developers

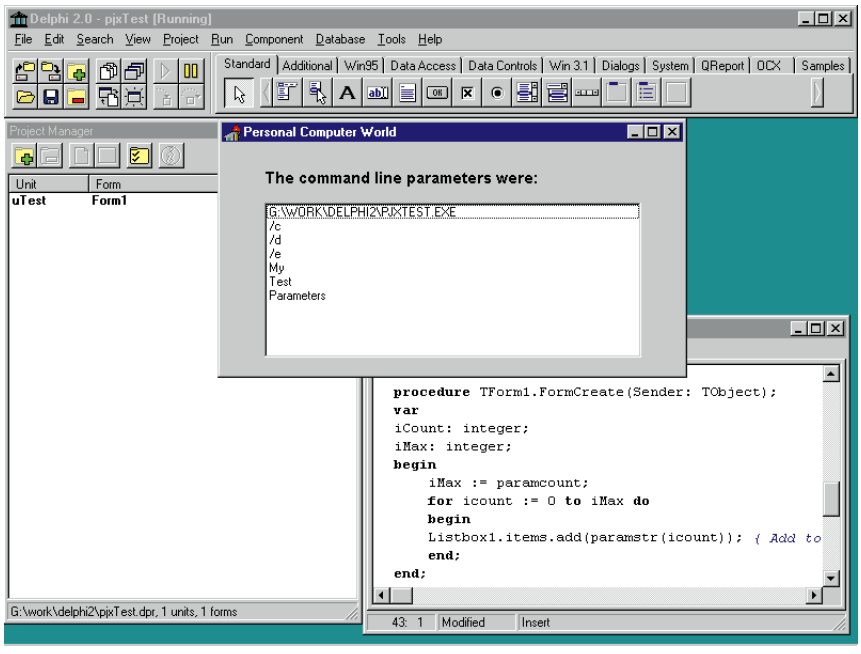
p318 ➢

Delphi tip: Detecting command line parameters

It is often useful to supply parameters to an application at startup. For example, if an application handles documents, then passing the name of a document as a parameter should run the application and open the document. Delphi has two functions to make this possible. ParamCount() returns the number of command line parameters, and ParamStr(Index: Integer) returns a string representing the parameter that corresponds to Index. ParamStr(0) always returns the application name with full path.

The following routine detects command line parameters and writes them to a log file:

```
var
iCount: integer;
iMax: integer;
F: textfile;

begin
  AssignFile(F, 'C:\TESTLOG.TXT');
  Rewrite(F);
  iMax := paramcount;
    for icount := 0 to iMax do
    begin
    Writeln(F, paramstr(icount)); {
                write to log }
    end;
  CloseFile(F);
end;
```



Andrew Jeffries must think so, since he asks: *"I am having a few problems writing 32-bit screensavers using Delphi 2.0, running Win95 and NT: How do you do a small preview in 95 without the configuration form always appearing? How do you implement security? How do you make the screensaver's name appear in NT and 95?"*

The problem with screensavers is that they are not well documented. The trusty Software Development Kits (SDKs) for the various Windows versions assume you will use C or C++, and that you will link your application with SCRNSAVE.LIB, a Microsoft-supplied library that holds the secrets of screensaver operation. Screensavers do not have to use SCRNSAVE.LIB, but avoiding it means extra work on the part of the programmer. Another snag is that screensavers work differently in each version of Windows.

Screensavers are executed by Windows in two ways: either when an interval of inactivity causes Windows to execute the screensaver, or when it is being configured in Control Panel. In Windows 95, screensavers have four modes of execution and these are selected by command line parameters:

- **Preview mode**. When you select the saver in Control Panel, Windows sends two parameters, /p HWND, to select preview mode and to pass the handle of the preview window.
- **Configuration mode**. When you click Settings, Windows sends a parameter, /c, to select configuration mode. The saver responds by presenting a configuration dialogue.
- **Password mode**. When you click to change the password, Windows sends two parameters, /a HWND, to select password mode and to pass the handle of the parent window for your password dialogue.
- **Start mode**. When you click Preview, or when the saver is called for real, Windows sends a parameter, /s, to select start mode.

So the answer to Andrew's first question is that the application should check the command line parameters to see whether it should draw in the preview window or present a configuration dialogue. See the tip panel (*above*) for how to detect parameters.

Screensaver security is treated in different ways by Windows 95 and Windows NT. Under Windows 95, most screensavers call the Windows Master Password Router. This is a DLL called MPR.DLL which exports password functions like PwdChangePassword. They are usually called via another DLL, PASSWORD.CPL, which works as an extension to the Control Panel. Neither of these libraries are fully documented in the Windows SDK, but some have worked out how to use them. The alternative is to implement your own password checking and throw up your own password dialogue when the saver is called in password mode.

Windows NT is different. Passwords for NT screensavers are the same as those used for logging on to Windows. The Control Panel marks a registry entry to indicate a secure screensaver:

```
HKEY_CURRENT_USER\Control
Panel\Desktop\ScreenSaverIsSecure
```

Finally, there is the matter of the description line. Confusingly, Microsoft has devised three ways of identifying this. Originally, it was the module description entry, which had to be of the form

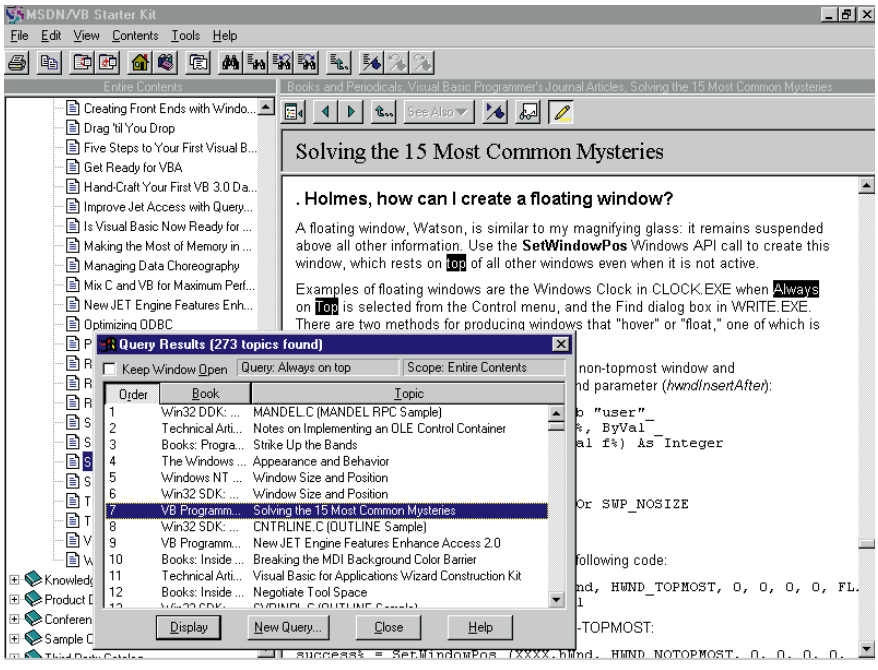
```
"SCRNSAVE : My Description."
```

Under Windows 95 and NT it became a resource string with an ID of 1 — and yes, Delphi can use standard Windows resources. This is the documented way; but actually, Windows 95 does not use it. It simply uses the long filename, less the .SCR extension. By the way, Windows will find any screensaver, identified by a .SCR extension, in the Windows or System folders, so at least installation is easy.

I've answered Andrew's questions, but I

p320 ➢

(Left) Free on our cover-mounted CD: the Microsoft Developer Network starter edition is a mine of VB and Windows information (Below) MSDN query results are displayed in order of likely relevance and clicking a title in the list displays the selected article



do not mean to suggest that writing screensavers is easy. The main problem is poor documentation, especially if you are not using Visual C++. A hunt around CompuServe or the web will throw up Delphi examples and help files created by other frustrated users.

■ Psst! Want Delphi cheap? Borland is bundling Delphi 1.0 with a book, *Teach Yourself Delphi in 21 Days*, and offering it for just £34.99 (plus VAT). A similar package for Delphi 2.0 costs around £69. Borland says the packs are aimed at "students, hobbyists and programming beginners".

VISUAL BASIC

The web is buzzing with talk of VB 5.0, now likely to be released early in 1997. There's even a web site devoted to VB 5.0 news and comments from anonymous beta testers. If the rumours are even half true, it looks like both performance and features will be hugely boosted.

In the meantime, Aaron Hodgson has contacted me with a question about Visual Basic: *"I am trying to write a terminal program using SAXCOM.VBX. The terminal works fine but I would now like to add an automatic logon sequence, where the logon details are read from an initialisation file when the remote computer on the other end of the modem prompts the user to log in. If the answer to my questions involves complex things like DLLs please explain, because I don't know the first thing about using DLL files with Visual Basic."*

Reading data from an initialisation file is not difficult. You can use API functions like GetPrivateProfileString, which uses a standard Windows .INI file. Searching the free MSDN sample on our cover CD-ROM will soon pull up an example. Alternatively, if you prefer native BASIC, there are the neglected statements like Open, Print# and Input#. These are well-documented in the Visual Basic manuals, with examples.

Microsoft Developer Network

By special agreement and arm-twisting, we've included the MSDN starter edition for Visual Basic on this month's cover-mounted CD-ROM. I get regular enquiries about MSDN, and now you can try it for yourself.

Although this is only a starter edition, there is 125Mb of documentation, tips and tricks included, so no-one need feel short-changed. It even includes two complete books: Petzold's classic *Programming Windows 3.1*, and BrockSchmidt's tome explaining OLE 2. Much of the information covers VB 3.0 as well as 4.0.

The best thing about MSDN is its fast searching. For example, you might want to know how to set a window to be always on top. Click search, enter "Always on top", and then Run Query. A moment later, MSDN presents 273 topics ordered by likely relevance. Article 7, "Solving the 15 most common mysteries", has a section explaining exactly how to do it.

A subscription to the full MSDN comes at several levels and prices. Information is available on the CD, or call Microsoft for details.

■ See next month's *PCW* for a review of the new Crystal Reports 5.0, and components from Sax software and Microhelp.

Cover Disk

Files from last month's *Hands On Visual Programming* were, unfortunately, left off the CD, but they can be found on this month's cover disc.

•PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address, or at **freer@cix.co.uk** or **www.cix.co.uk/~tim-anderson/**

Learn Borland Delphi in 21 Days £34.99 (plus VAT), or £69 (plus VAT) for the Delphi 2.0 version. Borland is on 01734 320022.
Visual J++ is by Charles Wood, ISBN 07615-0814-7. £32.99 from Computer Manuals, 0121 706 6000.
Microsoft Developer Network is on 0800 960279.

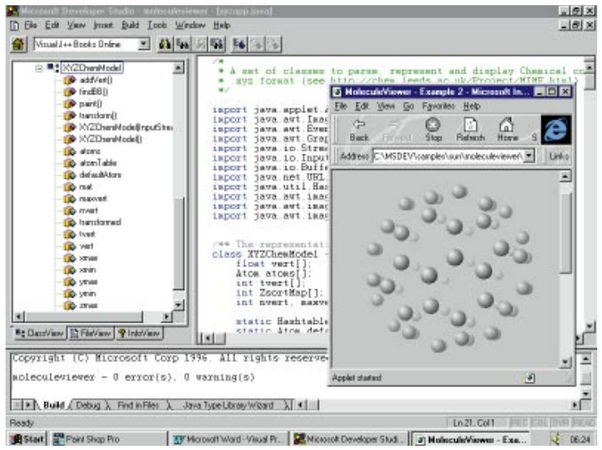
All together now...

We're all web developers now, says Tim Anderson, who previews Visual J++, deals with Delphi components, lists VB Script limitations, and serves up the Windows 95 system tray.

Should you care about the internet? Over hyped and under-powered, at least for those suffering modem connections, it would be easy to dismiss it as being of little relevance for most developers. Easy, but wrong.

Here are three reasons why, to keep your skills marketable, you have to be web-savvy:

Firstly, HTML is here to stay. It is ironic that Hyper-Text Mark-up Language, designed to add a few simple formatting options for web display, is evolving into the new standard for rich-text documents. The closest previous contender was RTF, or Rich Text Format, used internally by the Windows clipboard. But HTML does forms as well as documents, can host Java applets or, in its Microsoft incarnation, ActiveX controls, and is scriptable with JavaScript or VB Script. It blurs the boundary between document and application. The final consideration is the sheer momentum of cross-industry support. For anyone planning a new software project, an HTML front-end must be a strong contender, particularly for database applications.



Visual J++ looks just like its C++ partner and creates cross-platform Java applications

Secondly, users like browsers. Maybe the network computer will catch on, or maybe PCs running Windows will remain dominant. Either way, the browser is going to be the primary user interface. Once users discover they can manage files, run applications, get help and surf the web, all from the comfort of their browser, they will be reluctant to learn other kinds of interface. For developers, that means creating applications which work well in that context.

Thirdly, networks are intranets. Company intranets solve a lot of problems. Publishing documents, email, and database access are all comfortably handled by an intranet. If you buy a new server operating system, you are likely to find web-server software bundled with it, just as workstation software comes with a browser pre-installed. It is irresistible and companies that have not yet done so will inevitably install intranets alongside their Notes, Exchange, client-server or any other systems. Anyone developing software for use over a network should make it intranet-friendly.

So there are good reasons why software companies are falling over themselves to produce web software, and why you will see increasing coverage of web software tools in this column. The catch is that chaos is heading our way, with wars over standards, languages, objects and web servers. But we are all web developers now.

Visual J++ preview

A late beta of Microsoft's Java development tool has arrived in time for a brief preview. The package is hosted by the same Developer Studio used by Visual C++ and shares some of its tools. If you are comfortable with the Visual C++ environment, you will find the transition to Java easy. Each project can be viewed in a hierarchical class view, or file by file, and online documentation is fully integrated.

A clever touch is the resource wizard which converts compiled resources into equivalent Java code. There is still the problem, inherent to Java, that the Abstract Window Toolkit (AWT) Java class library does not support the range of controls available under Windows. Java projects can be either applets, hosted by a browser, or standalone applications which are executed from the command line using the supplied JVIEW tool.

Microsoft is licensed to produce the 32-bit Windows reference version of the Java virtual machine. The key point of interest is the integration between Java and COM, the object model behind OLE. This enables you to treat Java applets as COM objects and vice-versa. Visual J++ has a type library wizard which creates Java .CLASS files as an interface to ActiveX controls or OLE servers.

You can also expose Java interfaces as COM interfaces and use a tool called JavaReg to register the Java class as an OLE object. This means the Java class becomes accessible to OLE clients like Visual Basic or Delphi, as well as in web applications. Another interesting point is that when a Java applet is running in Internet Explorer, all its public methods and variables automatically become available to Visual Basic Script or JavaScript for scripting, as if the Java applet were an ActiveX control. The snag is that this integration only workd on platforms which implement COM, which essentially means Windows.

Visual J++ is a good Java development tool, whether or not you want the OLE features. Like Visual C++, it is not a visual environment in the same way as Visual Basic, Delphi or Optima. Borland's Latte promises something more along those lines. As proof that Microsoft is serious about Java, though, it is more than enough.

Visual Basic

Visual Basic and the System Tray

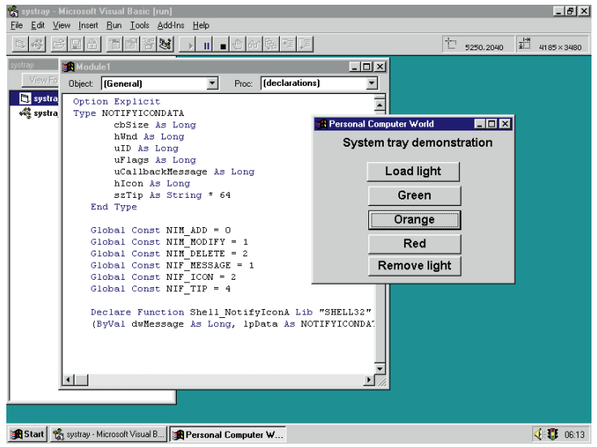
Sagar Shah writes: "*I recently moved up to Visual Basic Pro 4.0 (32-bit). While creating some small utility applications, I ran into a problem with the system tray on the taskbar. I cannot find any entries in the manual which tell me how to add to the system tray. I found the function Shell_NotifyIcon in the WIN32APTI.TXT file, and using this function I can add and delete a blank space but nothing more.*"

The system tray is a corner of the taskbar reserved for status display, in Windows 95 or NT 4.0. It is also called the taskbar notification area. Typically, a utility installs itself as an icon in the system tray, which automatically updates. For example, if you install a modem, a modem icon appears in the system tray when you go online and flashes when data is sent or received. When the mouse is over the icon, a tooltip shows more detailed information, in this case the number of bytes received. Double-clicking opens a dialogue of further options.

All this is done through the Shell_NotifyIcon function. The declaration in VB is:

```
Declare Function Shell_NotifyIconA
Lib "SHELL32" _
(ByVal dwMessage As Long,  lpData
As NOTIFYICONDATA) As Integer
```

The dwMessage parameter is one of three constants which tell the system to add, delete or modify an icon in the system tray. The second parameter points to a record type which can include an icon handle, a string for the tooltip, a window handle and an application-defined message identifier. The idea is that you define a message handler in your application which responds when the user clicks the icon. The message parameters indicate what type of

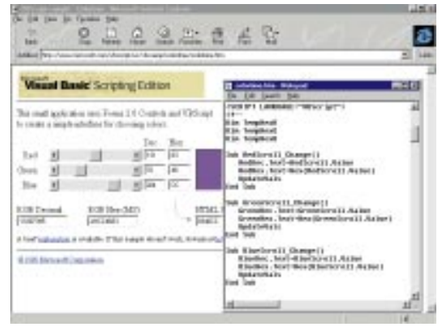


VB can easily control icons in the system tray, but responding to mouse clicks is more difficult

Visual Basic Script — what it does not do

Now that Microsoft Explorer 3.0 is around, VB Script has become useful, particularly on an intranet where you can ensure compatible browsers. Microsoft has also clarified its limitations, some of which are for security reasons while others are merely shortcomings of the language. Here are some of the things VB script cannot do:

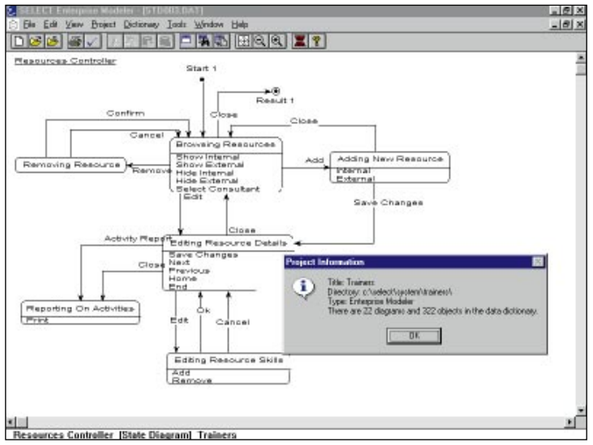
- No data access: data access from a web page needs either CGI scripting or special features of particular web servers.
- No debugging: you cannot even step through code.
- No control arrays.
- No classes.
- No OLE automation except the OLE interface to Internet Explorer itself.
- No file operations.
- No types, other than variants.
- No access to system objects like the printer or the clipboard.



Visual Basic Script is fine for scripting, but with major limitations

Many of these limitations can be overcome by using ActiveX controls, which have unrestricted access to the system. A rogue ActiveX could cause lots of problems, which is why system administrators are watching nervously to see if the digital signature scheme for verifying ActiveX controls is successful in preventing viruses.

p310



SELECT for Visual Basic Enterprise: not for the faint-hearted

mouse event has occurred. Your application window can be hidden so that it only pops up when needed.

Unfortunately, not all this functionality is available from Visual Basic. You can easily make an icon appear and set the text for the tooltip. But since VB has no way to intercept custom messages, you cannot make a dialogue appear when the user clicks the icon.

The way round this would be to use a control like MessageBlaster that adds this feature to Visual Basic. If you are content with more limited features, you can easily create an application like the example on our cover-mounted CD.

Note that you need to include an icon handle in the NOTIFYICONDATA record. You can obtain this in several ways: one is to use the icon or picture of a form or control; another is the LoadIcon API function; or there is the LoadResPicture function which works on icons stored in a resource file. The technique used in the example is to call LoadPicture to place an icon into an invisible image control, and then use its picture property to obtain an icon handle.

SELECT for Visual Basic

Bridging the gap between those who theorise about business object models and actual working systems is no trivial matter. SELECT is a set of tools based on object models developed by J Rumbaugh and Ivor Jacobson. There is a modelling tool, from which you can generate both SQL code and a set of Visual Basic forms and classes. There is also an automatic documentation feature that works with MS Word. SELECT needs the VB Enterprise edition, while other versions work with Forte and C++.

Unlike other tools which you can pick up and drop as required, committing to SELECT is almost a way of life. It replaces the free-and-easy VB style with a rigorous development process.

The best advice to those considering a system such as this is a careful evaluation procedure including full consultation with others actually using the system.

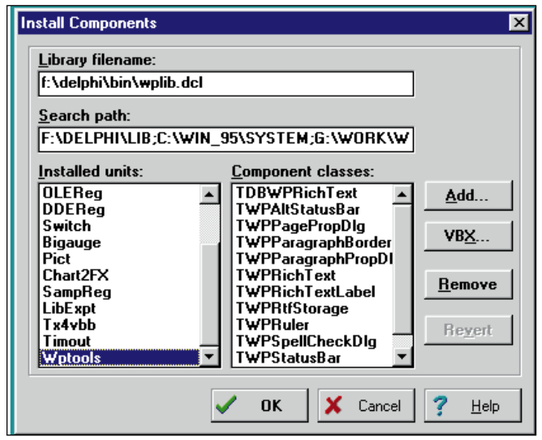
DELPHI

Using components in Delphi

When it comes to components, Delphi users have a difficult decision. The most obvious solution is to use VBX controls in Delphi 1.0, or OCX/ActiveX in Delphi 2.0. These component types are abundant, mainly thanks to the popularity of Visual Basic, and now that ActiveX plays a key role in Microsoft's internet strategy, you can expect them to proliferate.

Often, the component you want is only available in VBX or ActiveX form. The second advantage of these Microsoft standards is that they can be hosted by several different tools. For example, a VBX can be used by Delphi, Visual Basic and Visual C++, provided you use the 16-bit versions.

Unfortunately, these benefits are balanced by several problems, one of which is compatibility. Some VBX vendors assume that their controls will be hosted by Visual Basic and that not all their features work in Delphi. In particular, data-bound VBXs lose their data-aware functions in Delphi, if they work at all. Another problem is getting at certain VBX properties, such as string properties which Delphi converts to Pascal string types, cheerfully truncating them if needs be. OCX controls present a different set of problems, but since there are several different levels of OCX compliance, compatibility is by no means guaranteed. Finally, you cannot easily build either VBX or OCX components in Delphi. Borland has considered providing an OCX wizard to convert Delphi native components, but so far it has not emerged.



Delphi components are installed by rebuilding the component library. But which component type is best?

Delphi Component Options

Component type	Pros	Cons
VBX, ActiveX, OCX	• Widely available •Shared between applications	• Not always compatible • Cannot easily create in Delphi • Version control problems
VCL	• Best performance • Can create or customise in Delphi •No version control problems	• May not be available • Not shared between applications

p313 ➢

A book for visual programming

■ *Rapid Development* by Steve McConnell

Software projects are notorious for going wrong. The concept of rapid application development seems to offer a solution by using tools that dramatically cut programming time, but there are still plenty of problems.

Whereas his previous book, *Code Complete*, studied the detail of coding and debugging an application, this one analyses the whole development process, exploring common reasons for failure and offering tips for a successful strategy.

Really, the book is mis-titled. It is not only about rapid development, but any kind of software project. If your emerging application is suffering from feature-creep, unrealistic schedules, tools that do not work, problems integrating work from different team members, heavy overtime and low morale, then McConnell will tell you why, how to rescue the project, and how to avoid repeating history next time around.

Although the author does not focus on specific languages, there are some good observations about visual programming. The key advantage of products like Visual Basic and Delphi is reduced coding time, but there are associated risks. The main ones are over-estimated productivity savings, failure to scale well as the project expands, and the encouragement of sloppy programming. The answer is to be aware of the limitations of a particular tool and to allow time for working around them. Another key point is that the larger the project, the less time is spent on coding as opposed to other elements like designing and testing. Therefore, visual tools yield their biggest benefits on small projects.

There are plenty of case studies and examples in the book, although it is a shame that most are invented rather than real-life projects. It is repetitive in places and at times dispiriting, as the author tends to focus on how *not* to do things. But this is excellent reading for developers, especially those who work in a team.

compare with QuickReport for efficiency. Other leading component vendors have been slow to support Delphi but may find themselves losing sales to smaller upstarts as a result. A good example is WPTools, a VCL-implementation of a rich text edit control. This one works in 16-bit Delphi as well as Delphi 2.0 and is not just a wrapper for the Windows 95 common RTF control: it supports large documents, fonts, styles, images and hypertext links, and full source is available to registered customers. Look out for a full assessment in a future article.

Delphi 2.0 16-bit?

Borland is considering an update to 16-bit Delphi, and is using the web to survey developers about what features they would like and whether they would buy it. TI would prefer to see resources go to developing Delphi for 32-bit Windows. There is still a big market for 16-bit development, but I sense that the tide has turned. Companies have both NT 4.0 and Windows 95 from which to choose and web developments favour 32-bit Windows. It is too late for 16-bit Delphi 2.0, and Borland risks getting it wrong again.

A third-party has produced a full-featured rich text control as a native VCL component

To avoid the problems with VBX and ActiveX you can extend Delphi's Visual Component Library either by coding your own custom components or obtaining add-ons from a third party. Coding your own is fairly easy, although harder than straight Delphi programming. The advantage of native components is full compatibility and the most efficient interface between the component and the rest of your application, and therefore the best performance. The disadvantages are that the component can only be used in Delphi, and if you have several applications using the same component, inefficient use of disk space because the whole component is compiled into every executable you create. If this becomes a major problem, you can move parts of the code into dynamic linked libraries so it is shared between applications.

Frankly, in most projects a VCL component is preferable where available. Availability is the problem, but the situation is improving. Delphi 2.0 includes the QuickReport component which is vastly more efficient than wheeling in ReportSmith.

Crystal Reports 5.0 also comes with a VCL, although the huge Crystal DLLs do not

•PCW Contacts

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

Visual J++ (price not yet announced) from Microsoft 0345 002000

WP Tools is shareware from Julian Ziersch, 100744.2101@compuserve.com

SELECT VB Enterprise Edition £2,995. SELECT Software 01242 229700

Rapid Development by Steve McConnell, (Microsoft Press). £32.49 from Computer Manuals 0121 706 6000

Into the **Visual Age**

Tim Anderson looks at IBM's plans for VisualAge Basic, finds a new visual implementation of Prolog, and introduces new sections for Visual Basic and Delphi.

All programming is visual now. The quick riposte is that most programming tools are resolutely procedural with an array of visual tools to disguise the fact. But there's no doubt that the likes of Visual Basic and Delphi have won the argument about how to program. That leaves a difficult choice for a column like this one. With so many visual tools out there, should it become a pot-pourri of miscellaneous programming news and tips? Or should it revert to being product specific, dedicated to Visual Basic which remains the most popular Windows development tool? A further complication is that third-party components in the form of VBX or OCX/ActiveX controls can be hosted by a variety of different programming tools.

In response, Visual Programming Hands On has been expanded and will be divided into three parts. The first will cover visual programming generally, including components that are useful in a wide range of products. The other two sections will cover Visual Basic and Delphi respectively, so that users of the two most popular general-purpose visual languages will always find something specifically for them. Much of the material in this column is a direct response to your comments and queries, so please keep them coming to me, by email or at the usual *PCW* address.

IBM's new BASIC

At the time of writing, IBM is in open beta with its version of Basic, an addition to the VisualAge product family. The press release refers superciliously to Basic as a "scripting language", but nevertheless IBM's release is a great testimony to VB's influence. The final release date has not been announced, but will be before the end of the year. It is a cross-platform product, with versions initially available for OS/2 and Windows NT. Windows 95 compatibility will follow, and it will be possible to deploy Visual Age for Basic applications on AIX.

With Visual Age for Basic, IBM seems to have several goals in mind. One is to make OS/2 a more appealing platform, by introducing an enormously popular language and making it easy to convert existing Visual Basic applications. It is also part of IBM's attempt to establish its preferred object model, SOM, on the Windows platform. Visual Age for Basic will support SOM, OpenDoc and OCX. Finally, it is a tool for IBM's DB2 database, with integrated access using embedded SQL and the ability to create stored procedures and user-defined functions.

As a DB2 add-on or an OS/2 utility, Visual Age Basic looks likely to succeed, but whether it will challenge Visual Basic itself on the Windows platform looks more doubtful. Judging by the beta, system demands are as high or higher, with 24Mb RAM recommended for development. Like VB 4.0, it is an interpreted language and unlikely to win on performance. It is broadly compatible with Visual Basic, but that compatibility does not extend to data access code. On the plus side, IBM promises a proper implementation of inheritance and, should SOM catch on, Visual Age Basic will be very useful. Look out for a full review in due course.

Visual Prolog

Back in the seventies, a partnership between two Frenchmen, one a computer scientist and one a logician, produced a new language called Prolog (short for Programming in Logic). Unlike procedural languages, which give step-by-step instructions to the computer, Prolog does problem solving by inference and recursion. To give you a flavour, here's a complete program that looks up a telephone number:

```
PREDICATES
nondeterm tel_no(symbol,symbol)

CLAUSES
tel_no("Bill","0123 4567").
tel_no("Jane","0765 4321").

GOAL
tel_no("Jane", Number).

In this case the output is:

Number=0765 4321
1 solution
```

Prolog's particular strength is in artificial intelligence and expert systems. It would be a good choice for an application that assessed insurance risks or for a program to create timetables for schools, trains or airlines. In the late eighties, Prolog was marketed by Borland as Turbo Prolog, following which rights reverted to the Prolog Development Center (PDC). PDC has now come up with Visual Prolog, a graphical development environment for Windows (16 and 32-bit) and in due course for OS/2. Visual Prolog includes layout editors and Code Experts, which allow you to create a graphical interface by using drawing tools and responding to dialogs. It works by means of a set of Prolog extensions called the Visual Programming Interface, a framework for controlling a graphical interface.

PDC argues that Prolog's clarity and efficiency makes it not only a tool for building expert systems, but a challenge to more popular products like Delphi and Visual Basic. It compiles to native executables and performance is impressive. ODBC is supported for database work. A particularly nice feature is the integrated help authoring system, which makes it easy to create and edit online help from within the development environment. Nevertheless, the unfamiliar language combined with lack of support for VBX or OCX components, or OLE in any form, will ensure that Visual Prolog remains a niche product. For projects which lend themselves to a Prolog implementation, though, Visual Prolog is mightily impressive.

Visual Basic

Trouble with menus

VB programmer Ian Moss writes with a menu problem. "*I can add and remove items from indexed menus, no problem. What I want to do is create menus that have submenus. I am adding menu items that are divisions of a basketball league. Each division has teams associated with it. I read the division names from a database, and create the correct number of menu items. I want each division menu to have a sub menu containing the teams in that division.*"

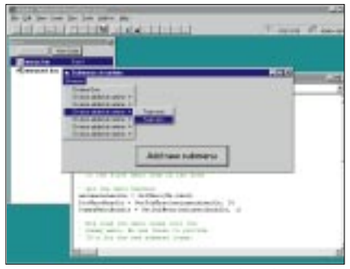
Here is a classic Visual Basic problem. Ian needs to create menus that have submenus, at runtime. VB's menu editor is a doddle to use. Creating menu items at runtime is easy using a control array and the Load command. But creating submenus at runtime is not in the book. It can be done, but only by trickery. It is another reason why serious VB programmers need Daniel Appleman's book, *Visual Basic Progammer's guide to the Windows API* (see review).

Using the Windows API, you can modify and add to the menus in a VB application.

Fig 1 VB Tip: Creating submenus at runtime

For this routine to work, you will need to use the VB menu editor to create a menu with two toplevel items. The first (Divisions) must have at least one sub-item, to make it a pop-up menu in API terms. The second (Dummy) must also have a sub-item, with an index value set, to make it a control array. I've named this mnuDummyArray. Finally, give the second toplevel item an empty string for a caption, and an enabled property of False. The user will never see the dummy menu.

Several API functions are included and these must be declared. The code below is for 32-bit Visual Basic, but with small amendments will work in 16-bit as well. Code to respond to clicks on the new menu items should be placed in the mnuDummyArray_Click event, using the Index parameter to detect which one was chosen.



You can create VB submenus at runtime, with a little help from the Windows API

```
Private Sub Command1 Click()

Dim mainmenuhandle As Long
Dim DivMenuHandle As Long
Dim DummyMenuHandle As Long
Dim NewMenuHandle As Long

Dim lRetVal As Long
Dim spareID As Long
Dim iCount As Integer

‘ this routine appends a new submenu
‘ to the first menu on the form

‘ get the menu handles
mainmenuhandle = GetMenu(Me.hwnd)
DivMenuHandle = GetSubMenu(mainmenuhandle, 0)
DummyMenuHandle = GetSubMenu(mainmenuhandle, 1)

‘ Now load two menu items into the
‘ dummy menu. We use these to provide
‘ ID’s for the new submenu items.

‘ count the existing items in the dummy submenu
iCount = GetMenuItemCount(DummyMenuHandle)

‘ load two new ones
Load Me!mnuDummyArray(iCount)
Me!mnuDummyArray(iCount).Caption = “Team one”

Load Me!mnuDummyArray(iCount + 1)
Me!mnuDummyArray(iCount + 1).Caption = “Team two”

‘ Create the new submenu
NewMenuHandle = CreatePopupMenu()

‘ Add two items to the submenu
spareID = GetMenuItemID(DummyMenuHandle, iCount)
lRetVal = AppendMenu(NewMenuHandle, MF ENABLED Or
MF STRING, spareID, “Team one”)

spareID = GetMenuItemID(DummyMenuHandle, iCount + 1)
lRetVal = AppendMenu(NewMenuHandle, MF ENABLED Or
MF STRING, spareID, “Team two”)

‘ Append the new submenu
lRetVal = AppendMenu(DivMenuHandle, MF ENABLED Or
MF POPUP, NewMenuHandle, “Division added at runtime”)

End Sub
```

p314 ➢

An entry with a submenu is not a normal menu item, but the top level of a pop-up menu, so you use the CreatePopupMenu function to return a handle to a new pop-up menu. Then AppendMenu is used both to add items to the submenu, and finally to add the pop-up menu to the existing menu structure.

The new menu will look pretty, but won't execute any code without further work. The problem is that VB creates menus with a two-stage process. When you design a menu, or change menu properties with VB code, you interact with an internal VB menu object. Visual Basic uses this internal object to generate the correct API calls that make the menu work. If you call the API directly, bypassing the internal VB object, VB doesn't know about the changes you have made.

When you click on a menu item, Windows sends a WM_COMMAND message to the application which includes a menu ID. This ID identifies the menu item, enabling the correct code to be executed. If you add a menu item using the API, VB will not recognise the ID, so the message sent when that item is clicked is ignored.

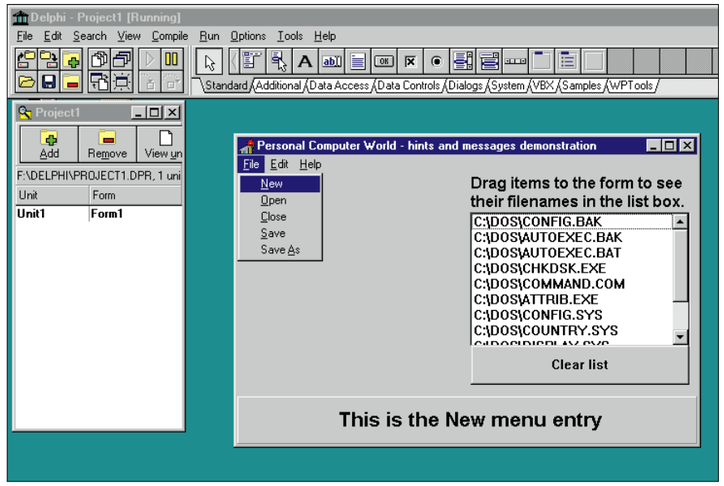
The workaround is to set up a dummy menu, where the toplevel item is disabled and has no caption, and which contains a control array. Note that the visible property must be True, otherwise the following tip will not work. When you need to add a menu item using the API, your code must first add an item to this control array. Then you can steal the ID for the new menu item, using the API call GetMenuItemID, and use it to create the new API menu item. When the user clicks on the menu you have created, VB is tricked into thinking that the item in the dummy menu has been selected, and will execute code in its click event. Fig 1 contains example code.

I must emphasise that this procedure is only necessary if you must create new submenus at runtime. Adding items to an existing menu or submenu is no problem. Another possibility is to create all your submenus at design time, and set their visible property to false, so that your code can reveal them as required. Finally, why not rethink the user interface completely? Ian's example application might be better served by an outline control, displaying divisions and teams in a tree view.

DELPHI

Delphi Gets the Message

One great thing about working with Delphi is how easy it is to trap Windows messages. Just to recap, much of Windows functionality is a result of system messages being sent to individual windows. For example, moving the mouse sends a WM_MOUSEMOVE message to a window.

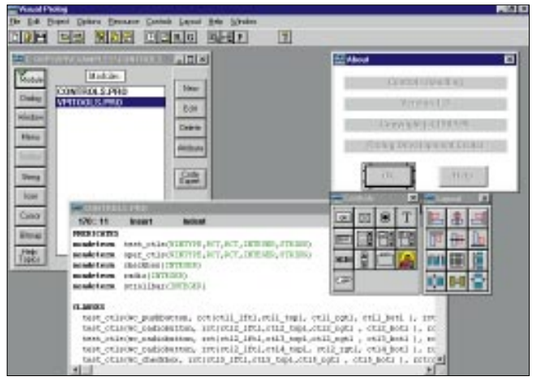


All done with messages: this Delphi application displays hints for the System menu, and accepts drag-and-drop files from Explorer or File Manager

Visual Basic Programmer's Guide to the Win32 API

Noted VB guru Daniel Appleman has issued an update to his popular API guide for Visual Basic users. This is no cursory update. The book has expanded by 500 pages and is more brick-like than ever. Even so, the author apologises for not including every Win32 API function. It is not his fault as the API is now so huge that to include everything would have made the book unmanageable. He correctly observes that once you know how the API ticks, it is not too hard to learn new functions and call them from VB.

Most serious Visual Basic developers will want this book. It accomplishes two things. First, it documents most API functions from a VB perspective, giving the correct declaration and explaining the particular benefits and pitfalls of each one. Second, there is masses on information on how Windows hangs together, including such topics as window handles, messages, co-ordinate systems and memory management. There's no other book like it, and Appleman does the job well.



Visual Prolog combines the Prolog language with a capable set of visual tools

I do have one nagging doubt, and that is why we need this book when Visual Basic should be powerful enough without it. The truth is, the deeper you get into the API from VB, the stronger the case for switching to a more suitable language such as C++ or Delphi. To get the best from VB, you need to be using it mostly within its natural limits, otherwise the benefits of RAD disappear under an avalanche of obscure code. The answer is to use this stuff with discretion, to solve problems that would otherwise leave you stalled. One example is optimisation. For instance, Appleman demonstrates a routine for searching listboxes that is five times faster than pure VB code. For users of your application that could make all the difference.

Sending, trapping, and creating custom messages are excellent techniques for creating powerful and flexible applications.

As an example, here's a couple of tips from Ian Briscoe (thanks for the tips, Ian - a book token is on its way). The first is for displaying hints for the System menu. The system menu does not appear in Delphi's menu editor, but you can still display hints by trapping the WM_MENUSELECT message. In the private section of a form declaration, add the following:

```
procedure SysMenuHint (var Message:
TWMMenuSelect); message
WM_MENUSELECT;
```

Note the message directive at the end of the declaration that tells Delphi this is a message handler. Fig 2 is the code for the procedure.

Ian adds, "Note that the menu selected is not the System menu. We call the inherited menu handler to allow Delphi to add its own hint functionality. We don't set the caption of the panel directly, but go through Delphi's own methods to display the hint, allowing you to still catch the OnHint event to add any additional coding."

The second tip is for trapping a drag-and-drop message from File Manager or Explorer. This is a neat trick that enables users to drag files into your application, for example to open documents in a text editor. First, add ShellAPI to the uses clause of the main form. Then declare the following message handler:

```
Procedure DragDrop (var Message:
TWMDropFiles); message
WM_DROPFILES;
```

Now in the FormCreate procedure add:

```
DragAcceptFiles(Form1.Handle,
True);
```

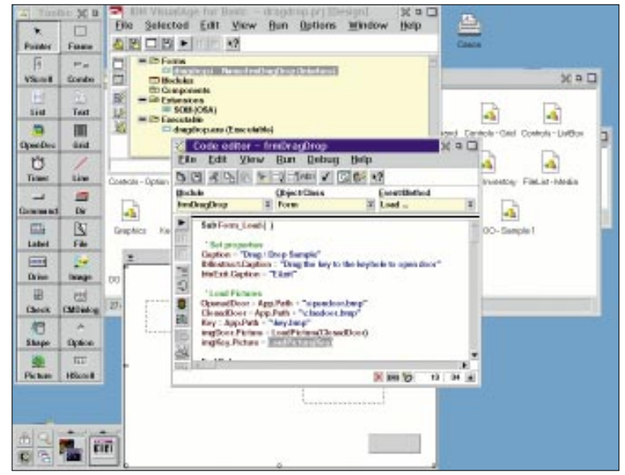
Fig 3 is the code for the DragDrop procedure. This example just displays the filenames in a listbox, but your code can do whatever you want with the files.

•PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address or at **freer@cix.compulink.co.uk**, or **http://www.compulink.co.uk/~tim-anderson/**

Visual Basic Programmers' Guide to the Win32 API is by Daniel Appleman. ISBN 1-56276-287-7. £46.99. Contact Prentice Hall, Tel. 01442 881900

Visual Prolog costs £477 from PDC UK, Tel. 01603 611291



VisualAge Basic brings easy application development to OS/2 at last

Fig 2 Code for message handler

```
procedure TForm1.SysMenuHint (var Message: TWMMenuSelect);
begin
if (Message.MenuFlag and MF_SYSMENU) = MF_SYSMENU then
begin
case Message.IDItem of
0: Application.Hint := '';
SC_CLOSE:
Application.Hint := 'Closes the window and quits the application';
SC_MAXIMIZE:
Application.Hint := 'Expands the windows to fill the screen';
{...etc. Look up the constants in WINAPI.HLP under WM_SYSCOMMAND}
else
Application.Hint := '';
end;
Message.Result := 0;
end
else
inherited;
end;
```

Fig 3 DragDrop code

```
procedure TForm1.DragDrop(var Message: TWMDropFiles);
var
i, numfiles: integer;
lpzFileName: PChar;
begin
numfiles := DragQueryFile(Message.Drop,Word(-1),nil,0);
ListBox1.Items.BeginUpdate;
lpzFileName := strAlloc(101);
for i := 0 to numfiles do
    begin
    strPCopy(lpzFilename,'');
    DragQueryfile (Message.Drop,i,lpzFileName,100);
    ListBox1.Items.Add (StrPas(lpzFileName));
    end;
strDispose(lpzFileName);
ListBox1.Items.EndUpdate;
DragFinish(Message.Drop);
Message.Result := 0;
end;
```

A break from the old **routine**

Tim Anderson **makes a splash with Visual Basic, and studies a slimline alternative to the Microsoft or Borland database engines.**

One of the keys to developing efficient, robust software, especially if you want to do so quickly, is to re-use code. Ways to do this include creating Delphi components, C++ classes, or using VBX or OCX controls in Visual Basic.

Dynamic Link Libraries (DLLs) are the foundation of Windows, and a great way to create functions that you can call from any programming language. You cannot create old-style DLLs with VB, but version 4.0 introduced OLE DLLs, allowing VB code to be called from other applications via OLE automation.

These are good ways to re-use code, but there is still a place for the oldest and crudest technique, which is cutting and pasting routines from one application to another. Programmers are lazy and will happily ransack old but working code to save time and avoid errors.

For example, a common requirement in VB database applications is to export a query as a .DBF table, the most universal format for mail merge, or transfer to other applications. JET's SQL supports a SELECT ... INTO clause that creates a new table from a query. If the database in question is an Access MDB, this only creates a new table in .MDB format. To get round this, I use this technique:

1. Output the query to a temporary table.
2. Copy the structure of the table to a new .DBF.
3. Copy the records in the temporary table to the .DBF.

This works well, and I have no intention of rewriting the code, which gets popped into applications as required. Only the second step takes more than a single SQL command, so this is wrapped in a re-usable function declared like this:

```
Sub CopyStructureToDBF(MDBName As
String, TableName As String, DBFPath
As String, DBFName As String)
```

It is vital that no paths or field names are hard-coded into the routine as this would wreck its re-usability.

The DIY solution

Once you have built up a library of routines, the next question is where to store them. Simplest is to have a directory full of .BAS files, but this is awkward to manage. It can also lead to the inefficient and unsafe strategy of including many unused routines in your project, for the sake of one or two that happen to be in the same module. A better solution is to write your own database application, storing each procedure in its own memo field.

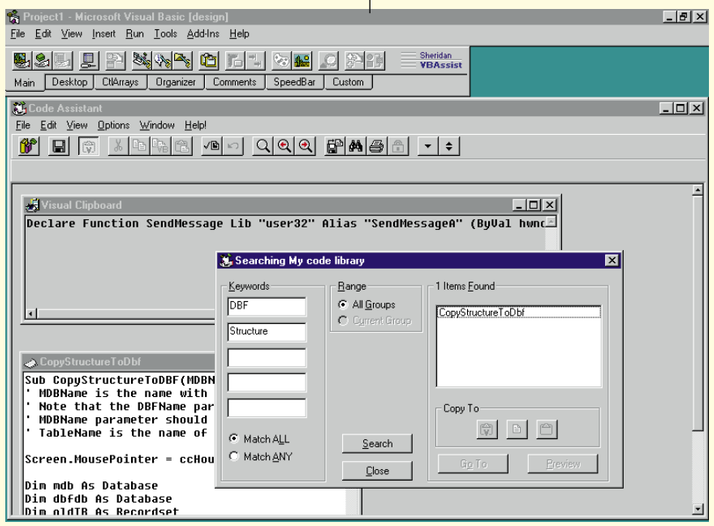
Alternatively, there are utilities that aim to make it easier to manage your code library. One is Sheridan's VB Assist, now at version 4.0a. VB Assist loads as an add-in, and includes Code Assistant. Code Assistant has two main elements. One is a visual clipboard, a text window to which clipboard output can be redirected. The other is a code database, called Code Librarian, which is actually a VB front-end to an MDB. You can create groups within which to store your routines, and add keywords for easy search and retrieval.

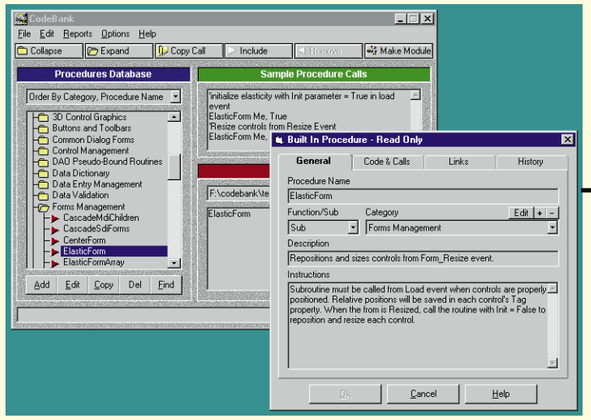
Code Librarian is a good idea, but it's not as well implemented as it should be. The way the database is structured suggests an outline tree for navigation, rather than the drop-down combos actually used. It is silly that keywords can be no more then ten characters long. You can edit code within the Assistant or Librarian, but it's not a good environment for coding, with no syntax highlighting or search-and-replace facility. But it's better than nothing.

CodeBank

Unlike Code Assistant, which is part of VB Assist, CodeBank is a separate product from Visual Components, best known for the Visual Developers Suite. Again it is a VB database application, but is a standalone program rather than an add-in. The idea is that all your re-usable routines are stored in the CodeBank database. When you want to make use of them, you ask CodeBank to create a new, empty .BAS module, and add the required routines. Finally, the generated module is added to your Visual Basic project in the normal way.

Part of Sheridan's VB Assist, Code Assistant lets you create libraries of code, and copy routines either direct to your application or to an intermediate clipboard





Codebank comes with a generous library of routines for slimline VB programming. No, there is not a tab control on this dialogue — it's all done with Basic

CodeBank has a tidy, effective interface. Procedures are shown in a tree, which can be sorted by category, author, name or type (procedure or function). Each routine can have substantial information stored with it, including short and long descriptions, an example of use, maintenance history, and links to any declarations or other routines that are required. Codebank is intelligent about these links: if a particular procedure makes use of a user-defined Type, the generated Basic module will include the declaration as well as the procedure itself.

The bonus is that CodeBank includes a library of 160 routines, with the emphasis on economy and performance. Many of them use VB code to emulate what might normally be done with a VBX or OCX control: for example, the outline used by CodeBank is drawn entirely using VB code. Another example is a procedure which shows text next to a control by printing directly to the controls' container, avoiding the need for a conventional label control. These routines are impressive, letting you create sophisticated graphic effects without the performance and size penalty of adding lots of components. Anyone interested in efficient VB coding will enjoy them.

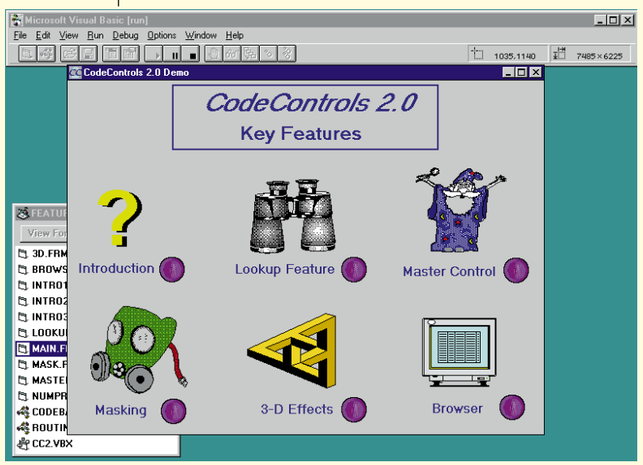
CodeBase 6.0

If the idea of distributing applications on a single floppy disk appeals to you, you will like CodeBase. Very small executables can be built in C, while VB or Delphi applications require a 500Kb runtime DLL, much smaller than either JET or the Borland Database Engine. Well-established in the xBase community, CodeBase is a C library for handling database tables in .DBF format, which are either FoxPro, Clipper or dBase IV compatible. Sequiter now provides versions for C++, Visual Basic and Delphi. The new version bundles the lot onto a single CD-ROM, which is convenient if you use more than one of these languages.

Other significant changes in version 6.0 are limited 32-bit support, the addition of client-server support via a new CodeBase database service application, and new transaction processing functions that will be useful in both standalone and client-server applications.

There are rough edges in this product. Although a 32-bit DLL is supplied for Visual Basic, the data-aware CodeControls are VBX only. An error in one of the main VB examples prevents it from running. Delphi support is currently only 16-bit, although a 32-bit DLL is available on request. Pascal documentation is more complete than in previous releases, but examples are in the form of short routines rather than a full demonstration application.

CodeBase can be integrated with your preferred visual tools, but not without some nitty-gritty coding



No effort has been made to create Delphi units or components to simplify use of CodeBase, which is a missed opportunity, bearing in mind the large number of migrants from Clipper, dBase and FoxPro now using Delphi. Successware, with its xBase product called Apollo, has done more to appeal to the Delphi community.

It is worth persevering, for the sake of fast performance on modest hardware, as long as you are willing to get your hands dirty with mysterious functions like "relate4createslave" and "code4initundo." While it is fine for both single and multi-user databases, it is harder to see the benefits for client-server work, unless you have an existing CodeBase system to upgrade. It is competing with many other advanced SQL-based systems, as well as another Sequiter product, the ODBC-compatible CodeSQL.

Code Complete makes a splash

Seasoned VB developers will know the story. A bemused user calls and says, "I tried to run your application. A message came up saying, 'Wrong version of SOMESTUFF.VBX', and then it quit." Windows is highly vulnerable to this kind of problem, and increasing use of OLE, which has its own myriad support libraries, will only make things worse.

Microhelp has a solution in the form of the Splash Wizard. From its name, you would think this is just a way of creating

fancy welcome screens, but this is secondary. The Splash Wizard creates a new executable which does comprehensive version-checking before launching your application. That way, problems can be identified before your application tries to load. Another possibility is to check for a valid user name and serial number. You can configure things so that your application can only run after the splash executable gives the OK.

Splash Wizard is a good idea, but I was not convinced by its implementation. It is fiddly to use, particularly since the wizard only operates from scratch. If you want to amend an existing splash executable, you have to tweak its resource file by hand, or by using a resource editor. Finally, in a simple test run, I tried out the Splash Wizard by deliberately deleting a .DLL needed by a VB application. The Wizard took a finger-tapping fifteen seconds to report the problem; the VB application on its own took two or three seconds.

Splash Wizard is an expert version checker, but can be slow

Code Complete comes with three other components. The Assistants automate the creation of common dialogues, message boxes, and allocating help IDs to VB controls, this last one being the most useful. Code Analyst will analyse and cross-reference Visual Basic projects, commenting on unused code and identifying deviations from standards you specify. For example, you can check that all modules include Option Explicit, or that error handling is enabled in all procedures.

If you have problems, the fourth component, AutoCoder, may help you out. A template-based system, it can automatically edit your code by adding error handlers for example. Another useful trick is to add temporary timing functions so that you can profile the application, discovering which routines are slowing down your software and need tweaking.

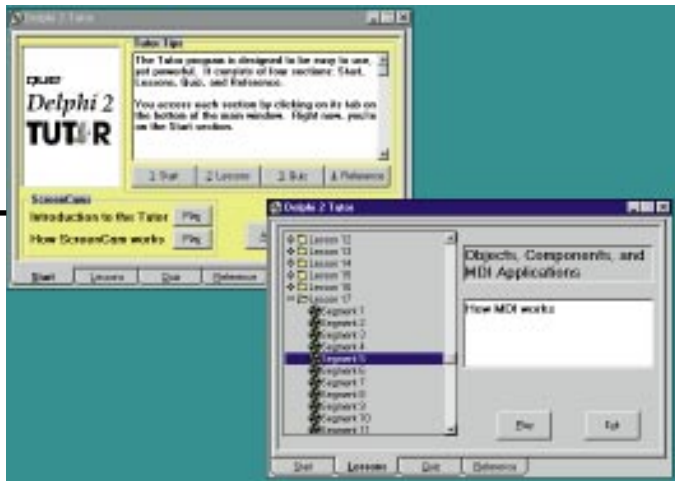
And finally...

Keep honing those Visual Basic skills. Microsoft is licensing the next version of the VBA engine to third-parties, so expect to see it in new versions of applications including Photoshop, AutoCAD and Visio.

Visual Programming: read all about it

Delphi 2 Tutor, by Mike McKelvy
Ironically, the software which runs this Delphi tutor is written in Visual Basic 4.0, assisted by Lotus ScreenCam. It is the opposite to *Delphi Unleashed*. Introductory and shallow, the excuse is that it is for complete beginners. The special feature is that each lesson has several screen demonstrations with explanatory voiceovers; seeing something done is certainly a help, but in this case it is not well implemented. The interface for the tutorial application is poor, a shame in a teaching tool, and the reference section is skimpy and inadequate. While Mike McKelvy's accompanying book has clear explanations of basic programming concepts, there is not enough information here to build real applications of any substance. A better approach would be to take the reader step-by-step through creating an example project. Video demonstrations are counter-productive unless they encourage hands-on experience as well.

Delphi 2 Unleashed, by Charles Calvert
The first edition of *Delphi Unleashed* established itself as one of the best titles for serious Delphi developers. The author works for Borland and is well placed to uncover Delphi's inner workings. This is no cosmetic rewrite: the new edition has over 1,400 pages, and more than half of this bulky volume is completely new. For example, you get 50 pages on multithreading, 250 pages on databases, 150 pages on OLE, and 200 pages on multimedia development. It is an enormously useful resource, clearly written, with sound explanations of both Object Pascal and the Windows API. The sheer amount of material makes it an intimidating volume, both physically and otherwise. Some will be glad to know how to create windows without using Delphi's Visual Component Library; others will wonder why we need to be told. Overall, not for the faint-hearted or beginners, but still a great companion to Delphi's inadequate manuals and online help.



Delphi 2 Tutor includes plenty of video sequences, but neither the presentation nor the content is inspiring

PCW Details

Contemporary Software
07000 422224
(VB Assist 4.0a, £135; Code Complete, £175) **Visual Components**
01892 834343 (Codebank, £99)
Highlander Software 0181 316 5001
(CodeBase 6.0, £225)

Books
Books from **Computer Manuals**
0121 706 6000
Delphi 2 Unleashed (Sams). Book and CD, £54.95 (inc VAT)
Delphi 2 Tutor (Que). Book and CD £46.99 (inc VAT)

Not just a **pretty face**

...that's Visual Developers Suite. **Tim Anderson checks it out, answers Delphi queries and solves a common Visual Basic problem.**

I do have misgivings about the hundreds of VBX and OCX controls on the market. It's not that they are no good: many are excellent and enable you to create a database manager, a web browser or a word processor in less time than it takes a C programmer to create a single "Hello world" window.

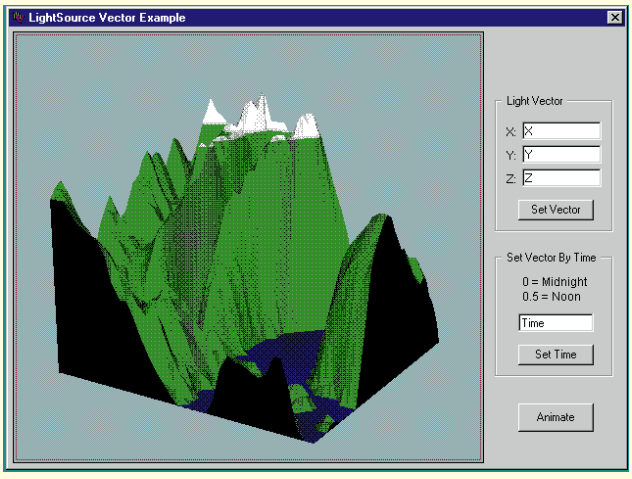
The problem is that every time you pop another component onto a Visual Basic form, your application grows more bloated and performance suffers. Canny developers will ask themselves, "Do I really need this control?" before committing to yet another OCX.

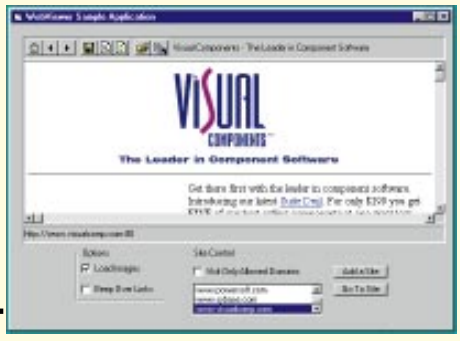
Farpoint's Tab Pro, now at version 2.0, is a case in point. Tabbed dialogues have become important in creating a clear, intuitive user interface and Tab Pro offers more than VB 4.0's native tab strip. It is supplied in every combination of 16-bit and 32-bit VBX, OCX and DLL, for use with virtually any Windows development tool.

The tab control is a container, unlike VB's tab strip which must be used in conjunction with another container like a picture box. Then there are more than 250 functions along with nearly as many properties which control the appearance of the tabs and which can look like a ring binder as well as a conventional tabbed dialogue. Tabs can be bound to a database to achieve a neat card-index style interface.

If you want a tabbed dialogue to look like a ring binder, Tab Pro is the obvious choice

Tab Pro is only around 300K and is well documented in two smart manuals. But do you really need it? Something like the Visual Developers' Suite, from Visual Components, makes better sense. No-one could accuse these controls of being merely decorative. Instead, this package includes five 16-bit and 32-bit OCX components, each of which is virtually a complete application in itself. There is the Formula One spreadsheet, First Impression charting control, Visual Speller, Visual Writer word processor control and, new in this version, WebViewer HTML control. The three most complex controls, for spreadsheet, charting and word processing, are among the best in their category.





First Impression, **(above)** *part of the Visual Developers Suite Deal, is an impressive charting component. WebViewer,* **(right)** *is new to the Visual Developers Suite but Microsoft's Internet Control Pack offers better functionality*

There are hesitations. Visual Writer is no longer indispensable now that a rich text control is part of 32-bit Windows and the WebViewer faces tough competition from Microsoft's freely-distributed Internet Control Pack. It is a shame that Visual Components has chosen to implement 16-bit OCXs, rather than the more widely supported VBX, for its 16-bit controls. But taken as a whole, the suite is excellent value and while these components will slow down your application, they also provide functionality that a VB developer could otherwise only dream about.

Do we still need Delphi?

Guy Robinson comments: "*As soon as Microsoft releases a compiler version of Visual Basic, Borland will have lost most of the advantage that Delphi currently possesses.*

With Microsoft controlling the operating system as well, Borland must ultimately lose the advantage. I found it surprising that Borland's Zack Urlocker (quoted in PCW*, May) wasn't more positive towards a cross-platform Delphi. Or is Java the company's intended cross-platform vehicle?*

I am an OS2 user, and if you talk to Mr Urlocker again you can tell him I would be one of the first to purchase Delphi for OS2 if it became available."

If you think cross-platform compatibility is important, and the rise of the Internet suggests that it is, then Java must be a more promising way forward for Borland than simply releasing versions of Delphi for other platforms. But I do not see Delphi being seriously threatened by a compiled Visual Basic. It is not just a matter of performance, it is the design and structure of Delphi that is richer and more elegant than Visual Basic. Another advantage is that Delphi is equally suitable for small utilities or major applications. VB has its own strengths, and a compiled version should address the performance issue, but Delphi will not lose its niche.

Combo Box defaults

Brendan Breen asks: "*I have a ComboBox in a dialogue. I want to set its style to DropDownList. When the dialogue is displayed I want to show a default value in the combo. But it always appears blank. I have tried all sorts of things but none of them work. Any ideas?*"

Writing to the Text or SelText property of a combobox does not work when its style is "csDropDownList". The solution is to write to the ItemIndex property. For example, you could put this into the dialogue's Show event:

```
ComboBox1.ItemIndex := 3;
```

Delphi 2.0 and OLE

Peter Harris queries Delphi's OLE capabilities: "*We are currently developing software using Borland's C++ (4.52) and OWL to develop a graphical interface to a specialised database. We were very interested in developing the front end in Delphi and purchased V2.0. However, we seem to have come across a fairly major limitation of Delphi2, in that it will not act as an OLE2 server and client. We need to be able to embed bitmaps and suchlike in our application windows, and also allow linking/embedding of our graphical stats results into other apps — specifically, word processors. We've been unable to find anything in the Delphi documentation about this. I wonder if you have come across this problem, or a way around it?*"

Delphi 2.0 will act as an OLE 2.0 server and client, but the server bits do not yet support embedding, at least not as implemented in the visual component library. On the client side, there is the TOleContainer which is to be found on the System tab of the component palette. For documentation, placing one of these on a form and then pressing F1 brings up all that Borland has seen fit to provide.

For example, you could create a link to a bitmap file with the following line of code:

Automating Delphi with OLE

Step by step, here's how to create a OLE automation server in Delphi:

1. Start a new application or DLL and save it as, for example, MYAPP.DPR. DLLs are in-process servers that run in the same address space as the calling application.

2. From Delphi's file menu, choose New and select Automation Object from the dialogue.

3. Enter a class name, for example MyObj. By default, Delphi will make the OLE class name the same as the application name, so the new OLE object will be MyApp.MyObj.

4. Choose the instancing. Internal is a rarely-used setting for OLE objects that are not available to other applications. Single instancing means that each instance of the server can only export one instance of the OLE object. Multiple instancing, which is required for DLLs, allows multiple instances of the OLE object.

5. Add OLEAuto to the uses clause in the project source. If it is a DLL, follow it with this section, observing case sensitivity:

```
exports
DllGetClassObject, DllCanUnloadNow,
DllRegisterServer,
DllUnregisterServer;
```

This is all you need. You don't need to add OLE objects and methods to the exports clause. Contrary to the documentation, you don't need to call Automation.ServerRegistration in the project source.

6. In the Automated section of the new OLE object, add the methods, properties and functions that are to be exposed, defining them in the implementation section in the normal way. There are limitations in terms of which types and declarations are allowed and normally these will be caught by the compiler if you try to use them. Here's an example type declaration:

```
type
  MyObj = class(TAutoObject)
  private
    { Private declarations }
    MyVar: integer;
    function GetMyProp: integer;
    procedure SetMyProp(iParm:
integer);
  automated
    { Automated declarations }
    function MyMethod(iParm:
integer): integer;
    property MyProp: integer read
GetMyProp write SetMyProp;
  end;
```

Note that you cannot access fields directly in an OLE object. You have to use property access methods.

7. Finally, the OLE automation object must be registered. Applications can be registered by running them with a /regserver parameter. You can register a DLL using Microsoft's REGSVR32.EXE utility, or failing that by calling the exported DllRegisterServer function. This need only be done once.

You can easily test the OLE object. Here is some example Visual Basic code:

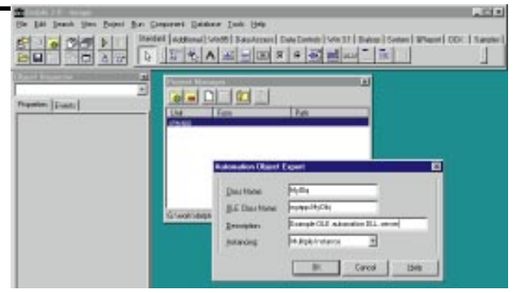
```
Dim myOLEobj As Object
Set myOLEobj =
CreateObject("myapp.myobj")
myOLEobj.myprop = 345
MsgBox "The property was set to: " &
str$(myOLEobj.myprop)
```

Why use OLE automation servers?

Performance of OLE servers is good, particularly in-process servers, but they are not as quick as standard DLLs. So why bother?

Firstly, because programming OLE objects is easy and intuitive, compared to ordinary DLLs which require case-sensitive function declarations.

Secondly, OLE objects bring with them the benefits of object-orientation, inheritance aside. Thirdly, OLE objects have a greater degree of language independence. Fourthly, as OLE progresses it should be possible to do things like remote automation using the objects you have developed.



Delphi's Automation Object Expert lets you enter the key characteristics of a new OLE automation object. Later, you have to register the object in the Windows registry

Visual Basic tip: detecting the IDE

How can you detect whether your Visual Basic application is running in the development environment or standalone? For example, you might want to enable extra debugging code when running in the IDE. Here are two ways to do it. The easy way is to use inspect App.Exename. App is a global object with several useful properties. For example, App.Path returns the directory from which the executable is run. App.Exename returns the name of the project, when running in the IDE, or the name of the executable when running standalone. If you give the project and the executable different names, then hey presto! you have an easy way to detect which is running.

If you would rather show off your Windows API skills, there is another method to use. All VB applications have a hidden parent window. In the IDE, this has a window class of ThunderMain, but in standalone executables the class is ThunderRTMain. Here's a function that exploits this difference to detect which is running:

```
' Declarations for 16-bit VB - amend for 32-bit.
Declare Function GetClassName% Lib "User" (ByVal hwnd%, ByVal lpClassName$,
ByVal nMaxCount%)
Declare Function GetWindowWord% Lib "User" (ByVal hwnd%, ByVal nIndex%)
Global Const GWW_HWNDPARENT = (-8)

Function isDev () As Integer
Dim ParentHwnd As Integer
Dim ParentClass As String
Dim iClassLen As Integer

ParentHwnd = GetWindowWord(form1.hWnd, GWW_HWNDPARENT)
ParentClass = String$(33, 32)
iClassLen = GetClassName(ParentHwnd, ParentClass, 32)
ParentClass = Left$(ParentClass, iClassLen)
If ParentClass = "ThunderMain" Then
isDev = True
Else
isDev = False
End If
End Function
```

```
OLEContainer1.CreateLinkToFile('C:\te
st.bmp',False);
```

OLE will then splutter and whir and the bitmap will be displayed. If the bitmap is later updated by another application, you can update it with:

```
OLEContainer1.UpdateObject;
```

In such a simple example, you could get similar results more efficiently using the LoadFromFile method of the picture in a standard image control, but the OLE approach has advantages. For instance, the OLE container will work with any OLE server on your system and supports things like in-place activation. You can save OLE objects to disk using TOLEContainer's SaveToStream method .

What about using Delphi as an OLE server, supplying information to display a graph or chart in a word processor document? Since this is not supported by the VCL, it is not easy to do in Delphi. The latest OWL or MFC-class libraries specifically support this OLE feature, so in this case C++ is a better option. What Delphi does support is OLE automation, so you could create an application that would supply the required data to any OLE automation client (see above; *Automating Delphi with OLE*).

If the client could also draw charts, as Microsoft Excel or Lotus WordPro can, then you could write code on the client side to extract the data and draw the graph. An insertable OLE object would be a better solution but an easy way to implement such a thing will have to wait for future versions of Delphi and Visual Basic. These will be able to create OLE controls; OLE automation objects with a visual interfa

PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address, or at **freer@cix.compulink.co.uk** or **http://www.compulink.co.uk/~tim-anderson/**

Contemporary Software 07000 422224 (FarPoint Tab Pro 2.0; £99 plus VAT).
Visual Components 01892 834343 (Visual Developers Suite; £235 plus VAT).

Right, said **thread**…

Tim Anderson explores threads in Delphi 2.0, picks up snippets from the VBITS conference, and answers your VB queries.

Microsoft's theme for 1996 is the Internet, which featured strongly at the recent London VBITS conference for Visual Basic developers.

At one session, the presenter rashly asked how many delegates were actually developing for the Internet. A scattering of hands were raised. Okay, how many plan to develop for the Internet? A few more hands. The message is that while tools vendors steam ahead with Internet products, the actual developers are mostly stuck in the old world of databases, accounts and local networks.

Another key question is how many developers have switched to 32-bit Windows. Microhelp's VB or OLE tools is a new product, available in 16- and 32-bit versions, and distributor Contemporary Software, exhibiting at VBITS, reports that sales in the first quarter of 1996 were 57 percent in favour of the 32-bit product — a one-off statistic but an indication that the move to Windows 95 and NT is finally happening.

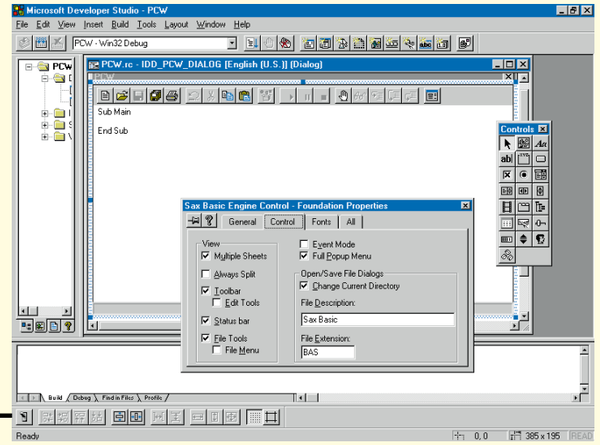
Those who did attend found a high standard of presentations, including API guru Daniel Appleman's demonstration of how to write a VB interface that runs as fast as C++. The answer is don't use controls, use VB's drawing methods instead. Of course, if you write VB applications like that you will be even less productive than your C++ counterpart. Even so, a point well made and a warning to go easy on controls, and especially VBX or OCX add-ons, if fast performance is a priority.

As expected, there are plenty of new Internet add-ons for Visual Basic and Visual C++. Microsoft's Internet Control Pack is a free download (beta at the time of writing) and contains OCX controls for integrating Web viewing, email, newsgroups and FTP file transfer into applications.

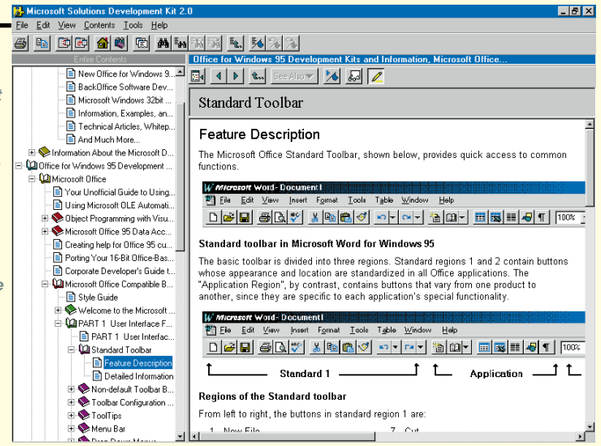
It's also been announced that Visual Basic 5.0, due out this year, will be able to create Active controls; another name for lightweight OCX components for Internet use.

The Internet again features in Visual C++ 4.1, an important update which adds MFC support for the Internet Information Server, Microsoft's Web server for Windows NT. A generous set of 12 third-party OLE controls has been added to Visual C++ 4.1, including Desaware's souped-up list box, the Sax Basic Engine for adding macro language support to your application, and Protoview's Interactive Diagramming Object for displaying data in the form of a diagram that can be visually modified by the user.

Finally, Microsoft has released the Solutions Development Kit, a CD which updates the Office Development Kit for those building applications with Office for Windows 95. For the latest news on Microsoft's tools, browse around the company's home page at http://www.microsoft.com.



Visual C++ version 4.1 comes with additional third-party controls including the Sax Basic Engine



The Solutions Development Kit is for high-level development with Microsoft Office. This page shows how to create an office-compatible toolbar

Books for Visual Programming

Wrox Press delivered one of the first books on Delphi 1.0, and now repeats the performance with *The Revolutionary Guide to Delphi 2.0*. Unfortunately, the trick is partly illusion, since much of the book covers 16-bit Delphi. This is a multi-author title aimed at those already competent with Delphi and attempts to cover every aspect of the package, making it a mixed bag. There are good chapters on debugging, component writing and the Windows API, along with skimpy coverage of the Borland Database Engine, ReportSmith, and issues specific to 32-bit Delphi. It would have been better to focus exclusively on Delphi 2.0 and cover fewer issues in greater depth. Nevertheless, the authors are knowledgeable and most Delphi developers will find plenty of valuable tips here.

● Charles Petzold's *Programming Windows 95* is a major new edition of a book revered by developers for its clear description of how Windows hangs together. The emphasis is on understanding Windows internals, starting with the creation and control of windows themselves and going on to include text and graphics, resources, memory management, input devices and dynamic link libraries. There are brand new chapters on the user interface, multitasking and multithreading and OLE, two of which are by co-author, Paul Yao. Given the huge complexity of Windows, Petzold is remarkably clear and concise. There is nothing here about Microsoft Foundation Classes, Visual Basic or Delphi, but simply an explanation of the Windows API with examples in C and occasionally C++. Highly recommended.

documentation for Delphi's multithreading support is some sketchy online help and one sample application. It is just another example of Delphi's rough-and-ready documentation, but is particularly disappointing given that this is unfamiliar territory for many developers. As it happens, Delphi's Visual Component Library includes a TThread object that simplifies multithreaded programming. What follows is a quick look at how it works.

Under Windows 95 and NT, each 32-bit running application is described as a "process" and has its own space in memory. A thread is an execution path within a process, sharing its memory but able to execute independently, with processor time allocated by the operating system. This means you can create applications which are more responsive, performing lengthy background tasks while remaining available to the user. Unfortunately, multithreading makes program design even more difficult and introduces new possibilities for bugs and conflicts: that is no reason to ignore threads, but it does suggest caution.

The example application is designed to cycle through 140,000 colour combinations, displaying the results in a panel control. Real-world applications would not do this, but might be rendering an image or downloading a file from the Internet, to name two common background tasks.

In order to spin this off as a separate thread, we derive a new thread class from TThread, declared as follows:

```
TPCWThread = class(TThread)
  private
    iRed: integer;
    iGreen: integer;
    iBlue: integer;
    thispanel: TPanel;
  protected
  constructor Create(panel: TPanel);
  procedure Execute; override;
  procedure UpdateColour;
  end;
```

All classes based on TThread must override the Execute method as this is the procedure which runs when the thread object is created. If you create the new thread class by choosing Thread Object from Delphi's object repository, the skeleton declaration will do this for you.

Keeping in step

The essence of multithreading is that at any time the operating system may switch processing time between one thread and another. This is no problem if the threads are truly independent, but what if they interact?

For example, TPCWThread needs to update a panel on a form. Other threads, including the main application thread, also have every right to update that panel. This is the reason for the warning comment that appears when you create a new Thread Object: "Important: Methods and properties of objects in VCL can only be used in a method called using Synchronize."

"Synchronize" is a TThread method which performs a vital function, letting you safely call VCL components such as Delphi forms and controls without conflicts. Synchronize takes a method name as its parameter.

In this example, there is an UpdateColour procedure which updates the panel, called from the main Execute method via Synchronize.

Calling the thread

When the user clicks the "Start a thread" button, the following code executes:

Parent problems

Chris John is contemplating a move to Visual Basic 4. He asks: "I have been thinking of upgrading from VB 3 to VB 4 but have been concerned about the problems of converting existing applications. I have written to Microsoft which has given me some comfort regarding conversion, but the company was rather non-specific when referring to API calls. Your reference to the API call SetWindowPos [in the April issue] has encouraged me. However, I have also used the call:

```
Declare Function SetParent% Lib "User" (ByVal H%, ByVal J%)
```

to enable me to add and remove member frames to, or from, an array of frames together with their contents (other control arrays) at runtime. Can you tell me what modifications to this call might be needed, or does version 4 provide for the addition of members to an array of frames together with their contents, which would do away with the need for an API call?"

Visual Basic 3.0 code should run fine in VB 4.0 16-bit version, but the move to 32-bits is problematic. For a start, VBX add-ons are not supported and although OCX versions are generally available, the transition is not always smooth.

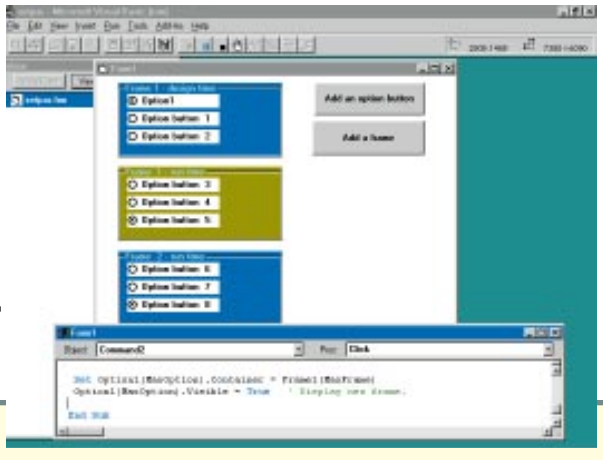
Next, although most API functions have a 32-bit equivalent, it is a different API and the declarations need changing. VB 4.0 comes with an API text viewer applet that lets you copy the declarations you need. In this case, the new declaration is:

```
Declare Function SetParent Lib "user32" (ByVal hWndChild
As Long, ByVal hWndNewParent As Long) As Long
```

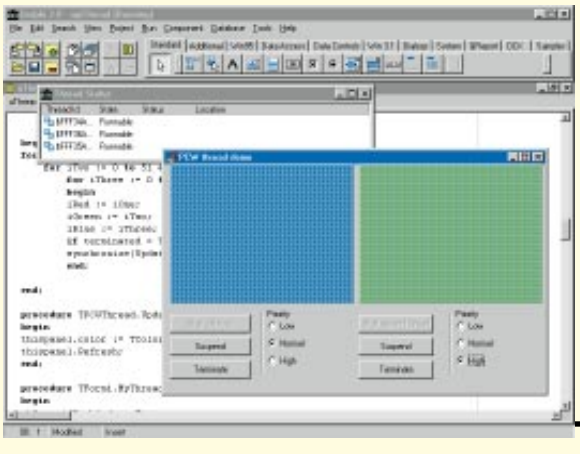
but the good news is that you probably don't need it.

In VB 3.0, although you can add new container controls like frames or picture boxes at runtime through a control array, the only way to add child controls to the new container is through the SetParent API call. VB 4.0 controls have a new Container property that overcomes the problem. For example, if you loaded a new frame at runtime, you could add an option button to it like this:

```
Load Option1(OptionIndex)
Set Option1(OptionIndex).Container = Frame1(FrameIndex)
Option1(OptionIndex).Visible = True   ' Display new button
```



This application shows how a control's Container property is used to add option buttons to a frame at runtime



You can check on Threads in a Delphi 2.0 application by showing the Thread Status window. In this demonstration, three are running, one for the main application thread and two TPCWThread objects. The user can control the priority of each thread while it is running. A thread can also be suspended or terminated

```
procedure TForm1.cbStart1Click(Sender: TObject);
begin
cbStart1.Enabled := False;
MyThread:= TPCWThread.create(Panel1);
Mythread.FreeOnTerminate := true;
Mythread.Priority := tpNormal;
rbNormal1.Checked := True;
Mythread.OnTerminate := MyThreadTerminate;
end;
```

To avoid creating two MyThread objects, the first line of code disables the button. Next, the thread object is created with the display panel passed to the constructor. The FreeOnTerminate property is set to true, which means the thread object is automatically destroyed when the thread stops running. Then, one of seven priority values is assigned, controlling how large a slice of processor action the thread receives. A corresponding radio button is checked.

Finally, a procedure is assigned to the Terminate event, so that the application can take appropriate action when the thread finishes its work. In this case, all the OnTerminate procedure has to do is re-enable the button.

There is no space here to print all the code but it can be found on our free, cover-mounted CD-ROM together with a compiled executable that anyone can run.

The finished application enables two TPCWThread objects to run side by side. Even with both running, the program remains responsive: the user can resize the window or move it around the screen. Each thread can be suspended and resumed, or heartlessly terminated before it finishes its task. The user can also control the priority that Windows gives to each thread.

All these are great benefits for certain types of application but this does not make it easy, so for serious multithreaded work, developers will need to look well beyond Delphi's sparse manuals.

PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address, or at **freer@cix.compulink.co.uk** or **http://www.compulink.co.uk/~tim-anderson/**

Visual C++ 4.1 and the **Solutions Development Kit CD** are available as part of a Microsoft subscription. The **Solutions Development Kit** will also be sold separately, price not yet available. Phone **0800 960279**

Books
All books available from **Computer Manuals 0121 706 6000** (prices incl VAT).
The Revolutionary Guide to Delphi 2 (Wrox Press). Book and CD £46.99
Programming Windows 95 (Microsoft Press). Book and CD £46.99

Ooh, you **Saxy thang!**

Build point-and-click access into your applications with Sax Webster, says Tim Anderson. There's widgets and Windows, too.

FORGET LAPTOPS AND MOBILE phones; the fashion accessory of the moment must be the personal Web site. Web sites are of no use unless they are visited, so why not build point-and-click access into the applications you distribute?

You can do this by calling an external application like Netscape or Internet Explorer, but Sax Software lets you go one better by building a customised browser right into the application.

The Webster control is a 32-bit browser OCX that drops directly into any compatible development tool, such as Visual Basic 4.0 or Visual C++ 4.0. With the rampant growth of the Internet and increasing corporate usage of Intranet networks, Sax Webster has turned up at just the right moment.

For example, online help might now mean dynamic information on a Web site, rather than the static file shipped with an application. Another option is to direct the hapless user to a site offering further products and services. HTML pages can be loaded from disk as well as from the Internet, so you could also use Webster as a multimedia browser.

Fig 1 Intercepting mailto

```
Private Sub Webster1_DoClickURL(SelectedURL As String, Cancel As Boolean)
If Left$(LCase$(SelectedURL), 7) = "mailto:" Then
' run MS Exchange, using file association
ShellExecute 0, "open", SelectedURL, "", "", 0
SelectedURL = ""
' stop Webster attempting to act on this command
Cancel = True
End If
End Sub
```

All done with Webster: VB 4.0 visits the PCW home page

Sax Webster is a complete application wrapped in a control. You can create a browser simply by dropping the Webster control onto a form in VB or Delphi. It claims to support HTML version 3.0, but Sax notes that "because 3.0 is not yet defined as a standard, it may differ from what Netscape, or some other 3.0 browser, supports."

Here is the problem with Webster, and ultimately with the Web itself: lack of tightly defined standards resulting in compatibility problems. It may not matter too much, since it would be foolish to use a Webster application as a replacement for Netscape or Internet Explorer. Webster makes better sense as a tool for accessing specific Web sites that are linked to the container application, so you can ensure the compatibility of those particular pages.

Some problems can also be overcome by writing code to intercept Webster events: for example, Webster does not support the mailto command that HTML uses to initiate an email message. The VB 4.0 code in *Fig 1* will intercept mailto and call whatever application is associated with that command in the Windows 95 registry.

Another useful feature is the GetContent method, which lets you read all or part of an HTML page into a variable.

Initially only available as a 32-bit OCX, Sax has now released a 16-bit OCX as well, but nothing yet for VB 3.0 or Delphi 1.0 diehards.

New-look Data Widgets

Sheridan's Data Widgets has long been one of the most popular Visual Basic add-ons, particularly since the VB 3.0 grid is so poor. The data-bound controls in VB 4.0 are better but still leave room for third-party enhancements. Version 2.0 brings the expected conversion to 16- and 32-bit OCX format, but with enhancements. Sheridan has taken the opportunity to restructure the Data Widgets using objects and collections, bringing it into line with other programmable OLE objects. This makes for more logical code and increases the programmer's control, the disadvantage being that code which worked with Data Widgets 1.0 will have to be extensively rewritten (*Fig 2*).

The actual Data Widgets controls are the same six as before: Data Grid, Data Combo, Data Dropdown, Data OptionSet, Data Command and the Enhanced Data Control. All are useful, but the Data Grid is the reason people buy this package. Its neatest trick is to link with a Data DropDown so that users can click on a grid cell and select values from a dropdown list bound to a field in another table.

Do you need Data Widgets? It depends on how you prefer to program. If you make extensive use of bound controls this bundle is all but indispensable, particularly if a data grid is a key part of the user interface. The data control in VB 4.0 is not compromised in the same way as VB 3.0's effort, so this is a perfectly sound approach. The cautionary note is that large OCX controls like these cause substantially slower loading of your VB application, and grids are often not the best way to present data to the user. The Data Grid also works well as an unbound virtual list control — a further enticement which may sway doubters.

Fig 2 On the button

To put a button in a DataGrid cell in version 1.0 of Data Widgets, use a ColBtn property:

```
SSDbGrid1.ColBtn(2) = True
```

which in Version 2.0 becomes

```
SSDbGrid1.Columns(2).Style = 1 ' edit button
```

Tools of the trade

Microhelp's OLE tools may have up-to-date OCX technology, yet the package conveys a dated impression: the main reason being that apart from their OCX conversion, many of the controls are little changed from earlier versions, right down to their description in the manual and the clunky example applications. OLE tools also slipped up during review when one of the genuinely new items, MhSubClass, failed to deliver. This is a message-trapping control that can catch Windows API messages and either kill them, or respond with a custom event and pass them on.

MhSubClass is fine for some purposes; for example, if you want to inspect WM_MENUSELECT messages in order to provide a help text as the mouse runs down a menu. But a common requirement is to trap a message and then write code to determine whether to kill it or pass it on. MhSubClass cannot do this, since the fate of the message has to be determined before the VB event is triggered. Rivals such as the MessageBlaster OCX have no such handicap.

Never mind the quality — with 54 separate controls, the bundle still rates as good value. MhCalendar is a data-aware

Top Tips: slim, dim and begin

- Speed VB's load time and slim your applications by stripping down AUTOLOAD.MAK (VB3) or AUTO32LD.VBP (VB4) to include only controls and references essential to every project.
- Avoid "Dim iA, iB as Integer". This code declares iA as a variant. Instead, use "Dim 1A as Integer, IB as Integer".
- In VB4, disable Compile on Demand (in Tools - Options - Advanced) to have the compiler check for syntax errors before a project runs.
- Your Delphi application can easily check for command-line parameters. ParamCount returns the number of parameters; ParamStr(0) returns the path and filename of the application; and ParamStr(n) returns the nth parameter up to ParamCount. For example:

```
procedure TForm1.Button1Click(Sender: TObject);
var
i: integer;

begin
for i := 0 to ParamCount do
  MessageDlg(ParamStr(i), mtInformation,
   [mbOk], 0);
end;
```

- If you are adding lines to a string control like a listbox or memo, or an outline component, use BeginUpdate to increase performance by preventing screen updates. For example,

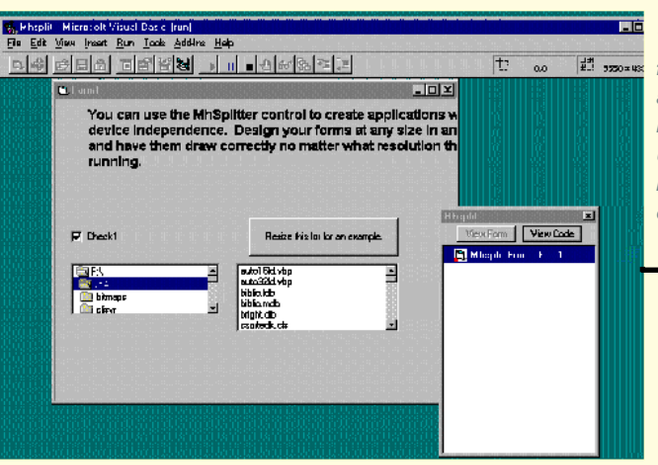
```
procedure TForm1.Button2Click(Sender: TObject);

begin
listbox1.items.beginupdate;
listbox1.items.add('One item');
listbox1.items.add('another item');
listbox1.items.endupdate;
end;
```

Fig 3 A little application to toggle with

Here are two functions to detect the status of the screensaver and to check its state. Note that to work in Windows 3.1, the declarations will need to be adapted.

```
Option Explicit
Declare Function SystemParametersInfo Lib "user32" Alias ➞
"SystemParametersInfoA" (ByVal uAction As Long, ➞
ByVal uParam As Long, lpvParam As Long, ByVal fuWinIni As Long) As Long
Public Const SPI_GETSCREENSAVEACTIVE = 16
Public Const SPI_SETSCREENSAVEACTIVE = 17

Function isActive() As Boolean
Dim lRetVal As Long
Dim pvParam As Long

lRetVal = SystemParametersInfo (SPI_GETSCREENSAVEACTIVE, 0, pvParam, 0)
If lRetVal = False Then
MsgBox "Call to SystemParametersInfo failed"
isActive = False
Exit Function
End If
If pvParam = False Then
isActive = False
Else
isActive = True
End If
End Function

Sub SetActive(bActive As Boolean)
Dim lRetVal As Long
Dim pvParam As Long
lRetVal = SystemParametersInfo ➞
(SPI_SETSCREENSAVEACTIVE, bActive, ByVal pvParam, 0)
If lRetVal = False Then
MsgBox "Call to SystemParametersInfo failed"
End If
End Sub
```

(Note: ➞ this symbol has been used where the code shown on the following line is a continuation, and should be entered as such. You'll find the complete code on this month's cover-mounted CD-ROM, together with versions for VB3 and Delphi).



The MhSplitter control from OLE Tools attempting resolution independence. Unfortunately, this text box does not always get correctly resized

calendar control. MhSplitter allows you to build resolution-independence into interfaces by automatically resizing controls within the container, albeit rather slowly. MhRealInput is a text box that improves on VB's masked edit control for working with real or currency values. And so it goes on, providing something of value for most VB projects.

Microhelp supplies two versions of these tools: OLE tools has 16- and 32-bit OCXs, while VB tools stays with the old VBX format. There are differences between the two. For example, the inadequate MhSubClass is OCX-only, while the clever MhOutOfBounds universal data binding control is VBX-only.

Finally, VB tools used to come with a version of Farpoint's Grid control, but that has now been dropped.

Screensaver settings: hacking into the Win95 API

"I've bought a new system with Windows 95 and VB 4.0. My computer has a WIn/TV card, and I wanted to write a program that would turn the screensaver off and on without having to go into the display properties tab.

How or where can I find out about the API calls necessary to change the screensaver settings? Is there a book which describes all the Win32 (and/or Win16) API calls?"

Mark Horton

Windows 3.1 introduced a handy function called SystemParametersInfo. This reads or sets numerous system parameters including the screensaver settings.

Fig 3 shows a small VB application for Windows 95 which toggles the screensaver on and off. The two key functions, IsActive and SetActive, work by calling SystemParametersInfo. The application checks the current state of the screensaver on loading, so that it can be restored on exit.

Another possibility is for your application to disable the screensaver whenever it has the focus. Windows activates the screensaver by sending a WM_SYSCOMMAND message with wParam set to SC_SCREENSAVE. By intercepting and killing this message, you prevent the screensaver from kicking in. Delphi programmers can trap messages easily but VB users will need an add-on like the MessageBlaster OCX.

Many problems such as this can only be solved using the Windows API. That, in turn, means having a good API reference and the starting point is the Windows SDK help file called WIN31WH.HLP for Windows 3.1, and WIN32.HLP for 32-bit Windows. Surprisingly, Visual Basic 4.0 comes with declarations for the 32-bit API but not the 20Mb help file.

An alternative is Daniel Appleman's book, *VB Programmer's Guide to the Windows API*, which provides what's needed for Windows 3.1 and is to be updated for Win32.

PCW Contacts

Tim Anderson welcomes your Visual Programming comments and tips. He can be contacted at the usual *PCW* address, or on **freer@cix.compulink.co.uk** or **http://www.compulink.co.uk/~tim-anderson/**

Contemporary Software supplies Sax Webster £110; Data Widgets 2.0 £99; OLE Tools £149; and VB Tools £99. Tel **01727 811999**